Joakim Sandqvist

Koonnin monitorointi

Metropolia Ammattikorkeakoulu
Insinööri (AMK)
Koulutusohjelma
Insinöörityö
Päivämäärä

Metropolia

Tiivistelmä

Tekijä Otsikko Sivumäärä Aika	Joakim Sandqvist Koonnin monitorointi 44 sivua + 0 liitettä 19.5.2012
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaajat	Yliopettaja Erja Nikunen Yliopettaja Auvo Häkkinen

Jatkuva integraatio tähtää projektityöskentelyn sekä ohjelmistotuotteen kehittämisen helpottamiseen. Ohjelmistotuote on tarkoitus saada valmiiksi mahdollisimman kivuttomasti ja kehitysprosessissa olisi hyvä törmätä mahdollisimman vähän kehittäjien ohjelmoimien ohjelmistomoduulien tai -komponenttien välisiin konflikteihin.

Jotta konfliktit ohjelmistomoduulien ja -komponenttien välillä huomattaisiin mahdollisimman nopeasti, tarvitaan tapa ilmoittaa konfliktin tapahtumisesta, eli koonnin epäonnistumisesta, kehitysryhmälle ja muille projektin osapuolille mahdollisimman nopeasti. Perinteinen tapa tämän tekemiseen on lähettää eri osapuolille sähköpostia. Tämä ei kuitenkaan ole paras eikä huonoin tapa, vaan yksi monista tavoista välittää informaatio koonnin tilasta kehitysryhmälle. Opinnäytetyössä tullaan esittelemään erilaisia ratkaisuja informatiivisen kehitysympäristön rakentamiseen.

Tämä opinnäytetyö tehtiin Metropolia Ammattikorkeakoulun Bulevardin toimipisteelle. Yksi opinnäytetyön tavoitteista oli suunnitella ja ohjelmoida reaaliajassa toimiva koontiympäristön tarkkailuun soveltuva ohjelmisto Ohjelmistotuotantoprojekti-kurssille. Sovellus voidaan luokitella osaksi kehitys- sekä työympäristöä.

Opinnäytetyössä kerrataan jatkuvaan integraation liittyvät perusasiat sekä perehdytään koontiympäristön tarkkailumenetelmiin. Tässä dokumentissa esitellään myös opinnäytetyön konkreettinen osa: Java-sovellus ”Koonta” ja esitellään kurssin uusi kehitysympäristö.

Työn lopputuloksena saatiin valmiiksi Java-sovellus Koonta, jonka tarkoitus on antaa ohjelmistokehitysryhmälle suoraa palautetta ohjelmistoprojektin tilasta. Ohjelmistossa ei ole vielä kaikkea asiakkaan haluamaa toiminnallisuutta, mutta ohjelmiston kehittämistä jatketaan opinnäytetyön valmistumisen jälkeen.

Avainsanat	jatkuva integraatio, koonti, Jenkins CI, Java, Agilefant

Metropolia

Abstract

Author(s) Title Number of Pages Date	Joakim Sandqvist Monitoring the Build Environment 44 pages + 0 appendices 19 May 2012
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Specialisation option	Software Engineering
Instructor(s)	Erja Nikunen, Principal Lecturer Auvo Häkkinen, Principal Lecturer

Continuous integration aims to ease project management and the software development process. The goal is to have a software development process that has as few component and module conflicts as possible.

For the development team to be able to react to conflicts swiftly, a convenient way to feedback the team is needed. Did the build fail? Who broke it? Feedback can be produced in the form of e-mail. This is not the best, nor is this the worst way to feedback the team. This is one of the many ways to notify the development team of a broken build. This document offers a look at some of the feedback methods and presents an example of an informative development environment.

The thesis was carried out for the Degree Programme of Information and Communications Technology of the Helsinki Metropolia University of Applied Sciences. The object was to design and program a piece of software for monitoring the build environment in real time. The piece of software is to be used by the students in the Software Development Project course to monitor their Jenkins CI build activities.

The thesis goes through the basics of continuous integration as well as introduces different build environment monitoring methods. The concrete part of the study, the Java based software: "Koonta", is presented and the work and development environments of the course in question are introduced as well.

Keywords	continuous integration, building, Jenkins CI, Agilefant, Java

Sisällys

Lyhenteet ja määritelmät

Agilefant Ketterien ohjelmistotuotantomenetelmien mukaisiin projekteihin soveltuva projektinhallintatyökalu.

Apache Ant Java-pohjainen koontityökalu. Katso luku 3.2.

Apache Maven
Koontityökalu. Katso luku 3.3.

CI Continuous Integration. Jatkuva integraatio.

CI-infrastruktuuri
Jatkuvan integraation toteutuksen perusrakenne. Muodostuu niistä palveluista ja rakenteista, jotka mahdollistavat jatkuvan integraation toiminnan.

CI-palvelu Jatkuva integraatio palvelu. Esim. koontipalvelimella sijaitseva jatkuvan integraation palvelinsovellus.

IDE Integrated Development Environment. Integrated Design Environment. Kaikki keskeiset sovelluskehitykseen liittyvät työkalut integroiva sovellus.

Jatkuva integraatio
Prosessi, jossa sovellusta käännetään, testataan ja integroidaan jatkuvasti kehityksen aikana.

Jenkins CI Integraatiopalvelinsovellus. Koontipalvelinsovellus.

Kehitysympäristöinfrastruktuuri
Kehitysympäristön perusrakenne. Muodostuu niistä palveluista ja rakenteista, jotka mahdollistavat kehitysympäristön toiminnan.

Koontityökalu
Koontiprosessin automatisoiva työkalu.

Koontipalvelin

Palvelin, jolla koonti suoritetaan.

Make Yksi vanhimmista ja suosituimmista koontityökaluista.

Projektioliomalli

Mavenin prosessikuvaus, tyypillisesti nimetty pom.xml:ksi.

Prosessikuvaus

Koontityökalulle syötettävä kuvaus, joka määrittelee koontiprosessin. Prosessikuvauksen sisältävää tiedostoa voidaan kutsua koontitiedostoksi tai koontikuvaustiedostoksi.

Scrum Ketterässä kehityksessä käytetty projektinhallintamenetelmä.

Subversion Versionhallintasovellus.

RSS-syöte RSS-feed. Tarkoittaa mm. Real Simple Syndication. Hyvä tapa levittää usein päivittyvää tietoa eri lähteisiin.

Versionhallintapalvelin

Versionhallintasovellusta, kuten Subversionia, ajava palvelin.

XF-laite Extreme Feedback Device. Extreme Notification Device. Laite joka antaa suoraa palautetta koonnin tilasta.

1 Johdanto

Tässä opinnäytetyössä esitellään Metropolia Ammattikorkeakoulun Ohjelmistotuontatoprojekti-kurssin käytössä oleva kehitysympäristö. Tämän kehitysympäristön esittelemisen työympäristöineen, palvelimineen ja palvelinsovelluksineen, on tarkoitus toimia suuntaa antavana materiaalina, niin pienten kuin suurienkin ohjelmistoprojektien koontiympäristöjen ja informatiivisten kehitysympäristöjen rakentamiseen.

Metropolia Ammattikorkeakoulun opiskelijat tekevät Ohjelmistotuotantoprojekti-kurssilla asiakkaan tilaaman ohjelmistotuotteen. Opiskelijat suunnittelevat ja ohjelmoivat tuotteen alusta loppuun. Opiskelijoilla on käytössään moderni kehitysympäristö, jota kehitetään entisestään koko ajan. Kehitysympäristö koostuu työtilojen laiteratkaisuista, työasemien sovelluksista sekä palvelin- ja palvelinsovellusinfrastruktuurista.

Tässä dokumentissa tullaan esittelemään opinnäytetyössä ohjelmoitu Metropolian Ohjelmistotuotantoprojekti-kurssin käyttöön tarkoitettu koontiympäristön monitorointi sovellus ”Koonta”. Sovelluksen tarkoitus on antaa kurssin työryhmille reaaliaikaista palautetta ohjelmistoprojektin koonnin tilasta.

Luvussa 2 tutustutaan ohjelmistotuotantoprojekti-kurssin sisältöön, kehitysympäristöön sekä työtiloihin. Tämän jälkeen luvussa 3 perehdytään ohjelmiston kokoamiseen käymällä läpi erilaisia koontiprosesseja eri koontityökaluilla.

Luvussa 4 kerrotaan jatkuvasta integraatiosta sekä koontiympäristöstä. Lisäksi esitellään sovellukset Jenkins CI sekä Agilefant, jotka liittyvät vahvasti lukujen 5 ja 6 aiheeseen koontiympäristön tarkkailu.

Lopuksi luvussa 6 esitellään opinnäytetyössä tehty Java-sovellus ”Koonta”. Tässä luvussa käydään läpi hieman sovelluksen ohjelmointiprosessia ja perehdytään sovelluksen toiminnallisuuteen sekä arkkitehtuuriin yksityiskohtaisesti.

2 Ohjelmistotuotantoprojekti -kurssi

Tässä luvussa perehdytään Metropolia Ammattikorkeakoulun Ohjelmistotuotantoprojekti-kurssiin. Luvun tarkoitus on antaa lukijalle hyvä yleiskuva kurssin sisällöstä, työtiloista sekä kehitysympäristöstä.

2.1 Ohjelmistotuotantoprojekti-kurssin yleiskuvaus

Opiskelijat pääsevät soveltamaan kurssilla useita aiemmilla tietotekniikan kursseilla opittuja tietoja ja taitoja. Työ on projektiluontoinen. Kuhunkin ryhmään kuuluu 4-6 opiskelijaa ja ryhmiä on yhteensä 2-4. Projekti kestää kahden periodin ajan, ja sen kokonaistyömäärä on noin 800-1200 työtuntia.

Ohjelmiston kehittämiseen käytetään Scrum-ketterää menetelmää. Scrumin työvaiheet koostuvat sprinteistä, joita tässä projektissa on viisi kappaletta. Jokaista sprinttiä edeltää aloituspalaveri, jossa sovitaan kyseisen sprintin tehtävät ja tavoite. Ydinroolit Scrumissa ovat tuotteen omistaja, tiimi sekä Scrum Master. Kurssin työryhmien Scrum Mastereina toimivat opettajat. Tiimit koostuvat kurssin työryhmistä ja tuotteen immateriaalioikeudet omistaa Metropolia Oy, joka puolestaan antaa tuotteen käyttöoikeudet projektiin osallistuneille tahoille. Tämä on toimintapa yleensä, mutta asiakkaasta riippuen tehdään erilaisia sopimuksia.

Jokainen kurssin ryhmä määrittelee, suunnittelee, toteuttaa, testaa sekä dokumentoi suurehkon ohjelmiston. Jokaisella projektilla on oma asiakas, joko oikea yritys tai opettaja, joka edustaa työn tilaajaa. Asiakas on käytettävissä kunkin sprintin aloituspalaverissa toiminnallisten vaatimusten määrittelyä varten. Kullakin ryhmällä on myös ohjaaja, jonka tehtävänä on auttaa työn käynnistämisessä sekä toimiminen Scrum Masterina. Ohjaajat eivät varsinaisesti osallistu tiimin työskentelyyn.

2.2 Ohjelmistotuotantoprojekti-kurssin kehitysympäristön yleiskuvaus

Kurssin viimeisimpiä kehitysympäristöparannuksia on Aleksi Lukkarisen opinnäytetyönään tekemä, ”...opetuskäyttöön soveltuva virtuaalikonepohjainen kehitysympäristö,

jonka avulla innovaatioprojekteja toteuttavat opiskelijat voivat työskennellä jatkuvan integraation ja käyttöönoton periaatteiden mukaisesti". [Lukkarinen 2011: tiivistelmä.]

Uusimmat parannukset kurssin työ- ja kehitysympäristöön ovat tähän opinnäytetyöhön kuuluva sovellus "Koonta", johon palataan vielä tämän dokumentin loppupuolella, sekä kurssin työtiloihin tehdyt uudet laitehankinnat.

Katselmus kehitysympäristöön aloitetaan esittelemällä kurssin uudet työtilat. Tämän jälkeen perehdytään kurssin palvelininfrastruktuuriin.

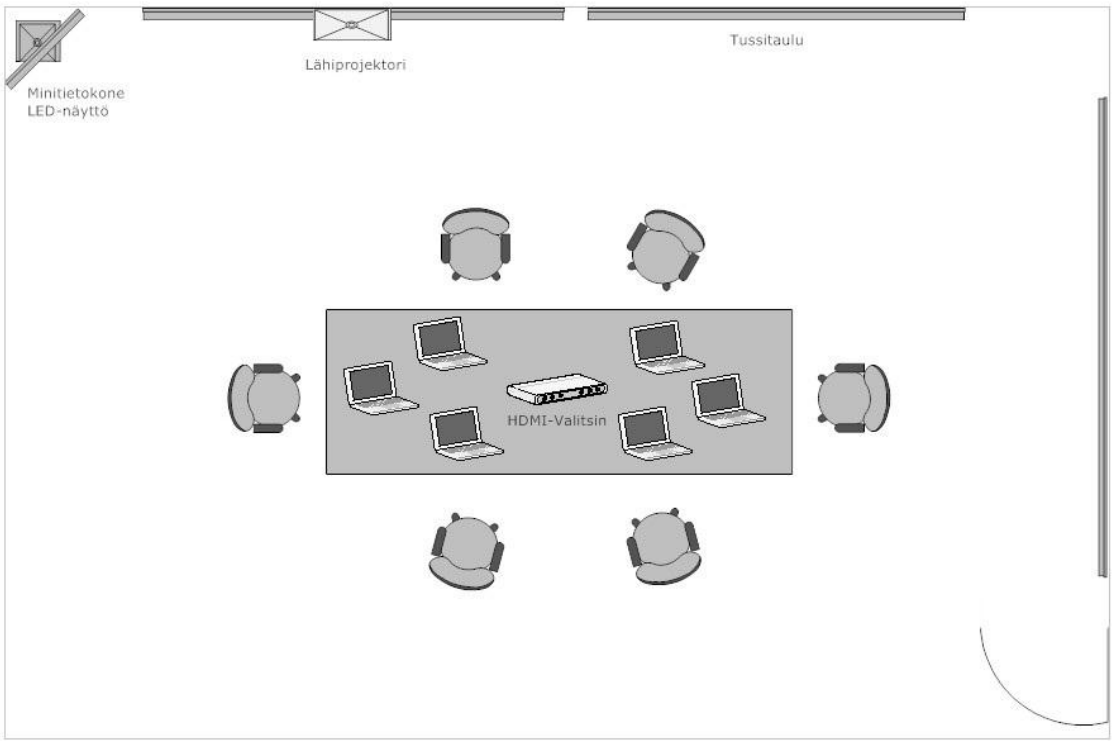
2.2.1 Kurssin työtilat

Työtilat ja viihtyvyys ovat olennainen osa ryhmäkeskeistä työntekoa. On tärkeää, että kehitysryhmällä on tarpeeksi tilaa työskennellä ja ryhmän sisäinen kommunikaatio toimii mahdollisimman helposti. Työtilat liittyvät myös olennaisesti koontiympäristön tarkkailuun, jota tässä opinnäytetyössä käsitellään myöhemmin.

Joulukuussa 2011 Metropolia teki uusia laitehankintoja Bulevardin toimipisteeseen Ohjelmistotuotantoprojekti-kurssin työtiloihin. Ensisijaisesti jokaisen kurssiryhmän työtiloihin oli saatava näyttö, jossa ajettaisiin tässä opinnäytetyössä tehtyä koontiympäristön tarkkailuun soveltuvaa ohjelmaa "Koonta".

Tässä luvussa käydään läpi kurssin työtiloihin hankitut laitteet ja esitellään työtilat kokonaisuudessaan mukaan lukien laitteet, jotka jäivät käyttöön vanhasta laitekokoonpanosta.

Kuvassa 1 on esitelty yhden ryhmän työtila. Kyseisen ryhmän työtilaan kuuluu kolme tussitaulua, joista yksi, lähiprojektorin alla oleva, toimii myös projektointipintana lähiprojektorille. Ryhmällä on käytössään kuusi kannettavaa tietokonetta. Tietokoneet sekä lähiprojektori ovat kiinni HDMI-valitsimessa. Kuvan vasemmassa ylänurkassa on minitietokone sekä näyttö. Minitietokone ja näyttö on varattu koontiympäristön tarkkailuun (luku 4). Ryhmä näkee laitteesta ohjelmistoprojektinsa tilan reaaliajassa ja pystyy reagoimaan projektin tilamuutoksiin nopeasti.



Kuva 1. Yhden ryhmän työtila

Yksi oleellisimmista uusista laitehankinnoista on lähiprojektori (kuva 2). Jokaisella kurssin ryhmällä on käytössään yksi lähiprojektori. Toisin kuin perinteinen projektori, lähiprojektori saadaan asetettua seinälle suoraan projektorointikohteen yläpuolelle. Täten se vie vähemmän tilaa kuin normaali projektori ja oli hyvä valinta uuteen laitekokoonpanoon.

Kuva 2. Hitachi CP-AW250NM

Työtiloihin hankittiin 2000 mm x 1200 mm tussitaulut, jotka toimivat lähiprojektorien piirtoalueina sekä vaihtoehtoisesti tussitauluina.

Jokaiselle ryhmälle hankittiin 27-tuumaiset LG E2770V -malliset LED-näytöt. Nämä näytöt on tarkoitettu vain koontiympäristön tarkkailua varten. Ne kestävät pitkäkestoista käyttöä ja voivat olla päällä yötä päivää ilman, että kuva palaa kiinni ruutuun. Näytöt on kiinnitetty seinään kolmi-nivelisillä Melicone Stile DR200 -mallisilla seinätelineillä, joiden avulla näyttöjen kaltevuutta ja sijaintia pystytään säätämään.

Ryhmille hankitut minitietokoneet (kuva 3) ovat kiinni yllä esitetyissä näytöissä. Minitietokoneita hallitaan langattomalla hiirellä sekä näppäimistöllä. Perinteisen pöytätietokoneen hankkimisen sijaan päädyttiin pienempään vaihtoehtoon, jotta työskentelytilaa jäisi enemmän.

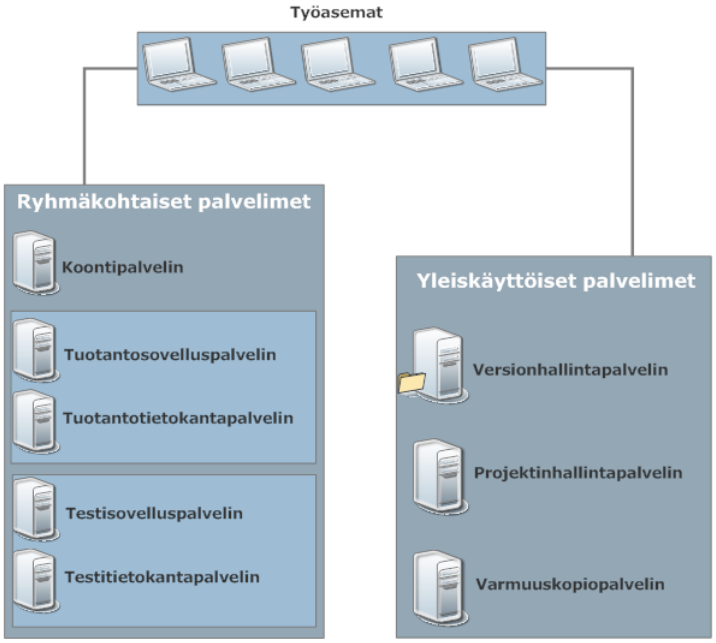
Kuva 3. HP8200 USDT -minitietokone

Tärkeä osa uutta laitekokoonpanoa on HDMI-valitsin (kuva 4). Valitsin on kytkettynä ryhmän lähiprojektoriin sekä ryhmän jäsenten käyttämiin kannettaviin tietokoneisiin. Valitsimella on tärkeä rooli ryhmien viikkopalavereissa, ryhmän päivittäisessä työskentelyssä sekä kehitysryhmän ja asiakkaan välisissä tapaamisissa. Valitsin mahdollistaa sujuvamman kommunikaation ryhmän jäsenten välillä. Kurssiryhmien kehittäjät saavat nappia painamalla oman kannettavan tietokoneensa näkymän lähiprojektoriin.

Kuva 4. Kramer VS-81H HDMI -valitsin

2.2.2 Kurssin palvelimet

Kurssin ryhmäkohtaiset palvelimet keväällä 2012 ovat koontipalvelin, testi- ja tuotantosovelluspalvelimet sekä testi- ja tuotantotietokantapalvelimet (kuva 5). Testipalvelimet toimivat ryhmän omana hiekkalaatikkona. Kehitysympäristön palvelininfrastruktuuri koostuu kokonaisuudessaan seuraavassa kuvassa esitellyistä palvelimista.



Kuva 5. Kehitysympäristön palvelininfrastruktuuri

Seuraavaksi käydään läpi palvelimet yksitellen. Kerrotaan, mitä sovelluksia kuhunkin palvelimeen on asennettu ja mitä ovat kunkin palvelimen tehtävät.

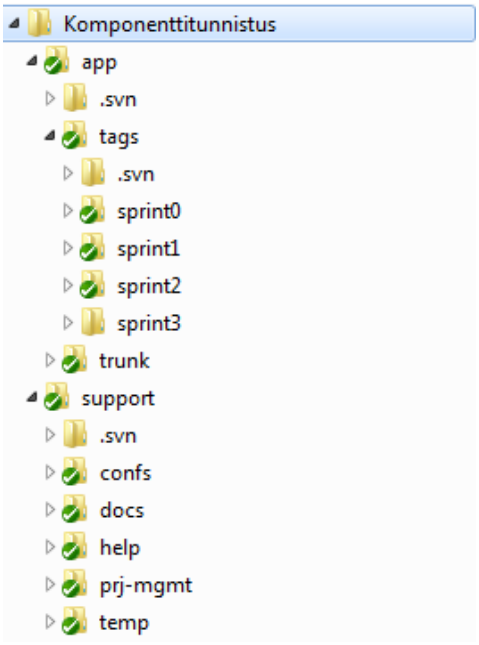
Palvelimista koontiympäristön (luvut 3.4, 4) kannalta oleellisimmassa roolissa on koontipalvelin. Jokaisella ryhmällä on oma koontipalvelimensa, jossa ajetaan Jenkins CI -sovellusta. Jenkins CI on vastuussa projektien automaattisesta koonnista ja hälyttää ryhmälle projektin koonnin epäonnistuessa. Jenkins CI:stä kerrotaan lisää luvussa 4.

Tuotanto- ja testipalvelimia on kaksi kappaletta jokaista ryhmää kohti. Tuotantoympäristöön kuuluvat tuotantosovelluspalvelin sekä tuotantotietokantapalvelin. Tuotantotietokantapalvelin sisältää ryhmäkohtaiset tietokannat. Tuotantosovelluspalvelin sisältää ryhmäkohtaiset palvelut, kuten Php:n ja Javan. Vastaavasti testiympäristöön kuuluvat testisovelluspalvelin ja testitietokantapalvelin, joilla on sama toiminnallisuus kuin tuotantoympäristön palvelimilla.

Ryhmien käytössä oleva testiympäristö mahdollistaa esimerkiksi palvelinkonfiguraatioiden riskittömän testaamisen ennen muutosten käyttöönottoa tuotantoympäristön puolella. Tuotanto -ja testipalvelimet ovat virtualisoituja. Palvelimien virtualisointi mahdollistaa palvelimen tilan tallennuksen. Palvelinten oletustilat (snapshot) voidaan halutessa ladata muistista, palvelimen korruptoituessa konfiguraatiotestaamisen yhteydessä.

Jokaisella ryhmällä on siis käytössään viisi ryhmäkohtaista palvelinta: koonti-, tuotantosovellus-, tuotantotietokanta-, testisovellus- sekä testitietokantapalvelin. Tämän lisäksi ryhmillä on käytössään kolme yleiskäyttöistä palvelinta, joista kerrotaan seuraavaksi. Nämä palvelimet ovat: versionhallinta-, projektinhallinta- sekä varmuuskopiopalvelin.

Versionhallintapalvelimella sijaitsevat ryhmien käytössä olevat kaksi tietovarastoa. Toista tietovarastoa käytetään lähdekoodin taltioimiseen ja toista tietovarastoa oheistuotteiden, kuten dokumentaation taltioimiseen. Opiskelijaryhmien tietovarastojen lisäksi palvelimella on innodev-tietovarasto ympäristön ylläpitoa ja kehitystä varten. Kuvassa 6 on esitelty esimerkki erään ohjelmistoprojektin tietovaraston hakemistorakenteesta.



Kuva 6. Esimerkki opiskelijaryhmän tietovaraston hakemistorakenteesta

Projektinhallintapalvelin sisältää Agilefant-sovelluksen, josta ajetaan yhtä instanssia kullekin kurssin ryhmälle. Agilefant on projektinhallintasovellus, joka soveltuu hyvin ketterän kehityksen tarpeisiin. Sitä käytetään ryhmän työtuntien merkkaamiseen ja projektin kehityksen valvomiseen. Agilefantista kerrotaan tarkemmin luvussa 4.

Kurssiryhmien käytössä olevalle varmuuskopiopalvelimelle on tarkoitus kopioida varmuuskopiota projektinhallinta- ja versionhallintapalvelimien tiloista. Varmuuskopiointi ei ole automatisoitua, jonka takia palvelimen käyttömäärä on jäänyt pieneksi.

3 Ohjelmiston kokoaminen

Ohjelmiston kokoaminen tarkoittaa yksinkertaisimmillaan ohjelmistokoodin kääntämistä ajettavaan muotoon. Jos ohjelmisto on iso ja sen rakenne on modulaarinen, voi kehittäjällä olla tarve jakaa ohjelmiston kokoaminen useisiin operaatioihin, joilla on keskinäisiä riippuvuuksia.

Tässä luvussa tutustutaan eri tapoihin toteuttaa ohjelmiston koonti. Ensin perehdytään makefileen ja tutustutaan ohjelmiston kokoamiseen makefile-esimerkkien avulla. Tä-

män jälkeen perehdytään modernimpiin koontisovelluksiin Ant ja Maven. Nämä luvut ovat tarpeellisia koontiprosessin ymmärtämisen kannalta.

3.1 Makefile

Makefile on tiedosto, jota Unix-komento Make käyttää ohjelman kokoamiseen. Makefile kuvaa ohjelman lähdekooditiedostojen väliset riippuvuudet ja sisältää ohjeet ohjelmiston kääntämiseen. Kääntämisen lisäksi makefileen voi sisällyttää mitä vain Unix-komentoja ja siitä voi käynnistää testejä.

Otetaan esimerkkitapaus, jossa C-kieliseen ohjelmistoon kuuluu 3 tiedostoa:

```
tiedosto1.c
tiedosto2.c
tiedosto1.h
```

Ilman makefileä ohjelma käännettäisiin Unix-komentoriviltä seuraavanlaisesti:

```
gcc –o esimerkki tiedosto1.c tiedosto2.c –I
```

Esimerkissä –I optio kertoo kääntäjälle, että sen tulee etsiä header-tiedostot työkansiosta. Gcc-kääntäjä kääntää tiedostot ja luo sovellusohjelman ”esimerkki”. Tämä lähestymistapa ohjelmiston kääntämiseen tulee vaivalloiseksi viimeistään siinä vaiheessa, kun projektiin lisätään tiedostoja; kääntämiskomento muuttuu ja se on haettava komentorivihistoriasta tai kirjoitettava uudestaan. Makefilen tekeminen ratkaisee tämän ongelman. [A Simple Makefile Tutorial.]

Esimerkki makefilestä näyttää tältä:

```
1. esimerkki: tiedosto1.c tiedosto2.c
2.     gcc –o esimerkki tiedosto1.c tiedosto2.c –I
```

Jos yllä olevat rivit kirjoitetaan ”Makefile”- tai ”makefile”-nimiseen tiedostoon ja kirjoitetaan komentoriville ”make”, niin suoritetaan makefileen kirjoitettu komento (rivi 2). Make ilman optioita suorittaa automaattisesti makefilen ensimmäisen kohteen (target), joka tässä tapauksessa on nimeltään ”esimerkki”. Rivillä 1 olevat tiedosto1.c ja tiedosto2.c kertoo makelle, että näiden tiedostojen olemassaolo vaaditaan kohteen suoritta-

miseen. Korostetaan, että make vaatii tabulaattorin käytön ennen jokaista komentoa (rivi 2). [A Simple Makefile Tutorial.]

```
1. KAANTAJA=gcc
2. OPTIOT=-I.
3.
4. esimerkki: tiedosto1.o tiedosto2.o
5.     $(CC) -o esimerkki tiedosto1.o tiedosto2.o $(OPTIOT)
```

Edellisen esimerkin riveille 1 ja 2 on lisätty muuttujat ”KAANTAJA” ja ”OPTIOT”. Näiden muuttujien muuttaminen vaikuttaa koko tiedostoon – tässä tapauksessa ainoastaan riviin 5.

```
1. KAANTAJA=gcc
2. OPTIOT=-I.
3.
4. all: esimerkki
5.
6. esimerkki: tiedosto1.o tiedosto2.o
7.     $(CC) -o esimerkki tiedosto1.o tiedosto2.o $(OPTIOT)
8.
9. tiedosto1.o: tiedosto1.c
10.    gcc -c tiedosto1.c
11.
12. tiedosto2.o: tiedosto2.c
13.    gcc -c tiedosto2.c
14.
15. clean:
16.    rm -rf *.o esimerkki
```

Edellinen esimerkki koostuu viidestä eri kohteesta.

Ensimmäinen kohde ”all” suoritetaan, kun Make ajetaan ilman optioita. Jotta ”all” voitaisiin suorittaa, pitää suorittaa ensin ”esimerkki” (rivi 6), koska ”all” on riippuvainen kohteesta ”esimerkki”.

Seuraavaa kohdetta ”esimerkki” ei voi kuitenkaan suorittaa, ennen kuin Make on tarkastanut tiedostojen ”tiedosto1.o” ja ”tiedosto2.o” aikaleimat; jos aikaleimat ovat vanhentuneet, eli tiedostot ovat muuttuneet, on ajettava kohteet ”tiedosto1.o” sekä ”tiedosto2.o”. Nämä kohteet ajettuaan voi Make siirtyä suorittamaan kohdetta ”esimerkki” ja tämän jälkeen kohdetta ”all”, jolla ei ole toteutusta.

Kohteella ”clean” ei ole riippuvuuksia. Tällaista kohdetta kutsutaan komentorivillä komennolla ”make clean”. Rivillä 16 oleva komento poistaa kaikki työhakemistossa sijaitsevat ”.o”-päätteiset tiedostot sekä sovellusohjelman: ”esimerkki”.

Viimeisin makefile-esimerkki on yksinkertainen ohjelmiston koonti, vaikka se vaikuttaa pelkästään ohjelman kääntämiseltä. Jos yllä olevaa esimerkkiä haluttaisiin kehittää lisää, voitaisiin siihen lisätä esimerkiksi kohde, joka hakee lähdekoodin tietovarastosta ja poistaa paikalliset lähdekoodit kovalevyltä ohjelman kääntämisen jälkeen.

3.2 Ant

Luvussa 3.1 tutustuttiin makefileen, joka on Maken koontikuvaustiedosto. Tässä luvussa perehdytään koontityökalu Antin käyttämään Xml-muotoiseen koontikuvaukseen.

Apache Ant on Java kirjasto- ja komentorivityökalu, jonka tehtävänä on ajaa prosesseja jotka on kuvattu koontitiedostoissa ”kohteina”, joilla on keskinäisiä riippuvaisuuksia. Ant on pääasiallisesti Java-sovellusten koontisovellus, mutta sitä voidaan käyttää myös C -ja C++ -ohjelmistojen kokoamiseen. Tässä luvussa on esimerkki Java-ohjelmiston koonnista.

Antin käyttäjät voivat kehittää omia ”antlibs”-kirjastoja, joihin sisältyy omia tehtäviä (tasks) ja tyyppejä (types). Käyttäjille tarjotaan myös suuri määrä valmiiksi tehtyjä kaupallisia tai avoimia ”antlibs” kirjastoja. [Apache Ant: Welcome.]

Ant-koontitiedosto koostuu neljästä eri elementistä: *projects*, *targets*, *tasks* ja *properties*.

Koontitiedostossa kuvatulla projektilla on kolme attribuuttia: *name*, *default* sekä *basedir*. *Name* kuvaa projektin nimeä ja eikä sitä ole pakko määritellä. *Default* kuvaa koontitiedoston oletuskohdetta (default target), jota kutsutaan, kun kohdetta ei erikseen anneta kokoamiskomennossa. *Basedir* määrittelee projektin työhakemiston, jota käytetään hakemistopolkujen selvittämiseksi. Tämäkään attribuutti ei ole pakollinen. Jos *basedir*-atribuuttia ei määritellä, käytetään työhakemistona samaa hakemistoa, jossa koontitiedosto sijaitsee.

Kohteilla (targets) on riippuvuuksia muista kohteista. Kuten makefilessä, voidaan kohteiden välille rakentaa riippuvuuksia ja luoda ohjelmiston kokoamisen kannalta järkevä riippuvaisuuksien verkosto.

Tehtävät (tasks) ovat suoritettavia koodirivejä. Tehtävillä voi olla useita attribuutteja, kuten tulevasta esimerkissä nähdään.

Ominaisuuksilla (properties) voidaan antaa oikoteitä (shortcuts) komentoihin tai merkkijonoihin, joita käytetään usein koontitiedostossa. Ominaisuudella on nimi- sekä arvo-attribuutti. [Apache Ant: Using Apache Ant.]

```
1.  <project name="MyProject" default="dist" basedir=".">
2.    <description>
3.       Yksinkertainen koontikuvaustiedosto
4.    </description>
5.   <!--Asetetaan julkiset ominaisuudet -->
6.   <property name="src" location="src"/>
7.   <property name="build" location="build"/>
8.   <property name="dist"  location="dist"/>
9.
10.  <target name="init">
11.   <!—Luodaan aikaleima -->
12.   <tstamp/>
13.   <!—Luodaan hakemistorakenne “compile” tehtävälle -->
14.   <mkdir dir="${build}"/>
15.  </target>
16.
17.  <target name="compile" depends="init">
18.   <!--Käännetään Java-koodi ${src}:stä into ${build}:iin -->
19.   <javac srcdir="${src}" destdir="${build}"/>
20.  </target>
21.
22.  <target name="dist" depends="compile"
23.       description="Generoidaan distribuutio" >
24.   <!--Luodaan distribuutio kansio -->
25.   <mkdir dir="${dist}/lib"/>
26.
27.   <!-- Laitetaan ${build} sisältö MyProject-${DSTAMP}.jarriin -->
28.   <jar jarfile="${dist}/lib/MyProject ${DSTAMP}.jar" basedir="${build}"/>
29.  </target>
30.
31.  <target name="clean"
32.       description="clean up" >
33.   <!—Poistetaan ${build} ja ${dist} hakemistot -->
34.   <delete dir="${build}"/>
```

```
35.     <delete dir="${dist}"/>
36.   </target>
37. </project>
```

[Apache Ant: Using Apache Ant, Example Buildfile]

Edelliseen koontikuvaukseen on määritelty projekti, jonka nimi on “MyProject”. Kun tutustutaan koonnin kuvaukseen huomataan, että koontitiedoston projektin oletuskohteella ”dist” on riippuvuus kohteeseen ”compile”, jolla on riippuvuus kohteeseen ”init”. Initillä ei ole riippuvaisuuksia. Täten ensimmäinen koonnin yhteydessä ajettava kohde onkin ”init” eikä ”dist”, vaikka ”dist” on koontikuvauksen oletuskohde. Voidaan sanoa, että koonnin päämäärä on suorittaa kohde ”dist” ja päämäärän saavuttamiseksi on suoritettava kohteita, joista tämä on riippuvainen. Riippuvuudet toimivat samalla periaatteella kuin aikaisemmissa makefile-esimerkeissä.

Ohjelmiston kokoamisen yhteydessä halutaan edellisessä esimerkissä saada ohjelmisto jakeluun. Jotta ohjelmisto saataisiin jakeluun, on se ensin käännettävä ja sitä ennen on luotava ohjelmiston hakemistorakenne.

Jos halutaan kehittää esimerkin koontikuvausta, voidaan esimerkiksi suorittaa verkkosivujen päivitystehtävä yhden kohteen sisällä ja asettaa se riippuvaiseksi kohteesta ”dist”. Näin ohjelmasta saadaan päivitetty versio jakeluun ja verkkosivuille tulee tästä automaattinen ilmoitus ohjelmiston käyttäjille.

3.3 Maven

Luvussa 3.2 tutustuttiin Anttiin, joka on natiivi koontityökalu. Seuraavaksi tutustutaan koonti- ja projektinhallintasovellukseen Maveniin, jolla voidaan kuvata koontiprosessin lisäksi myös projektin rakenne. Maven rohkaisee käyttämään parhaita käytäntöjä (best practises), joita käyttämällä ohjelmistoprojektista toiseen siirtyessä, oppimiskynnys uuden koontikuvauksen ymmärtämiseen pienenee. [Maven vs Ant or Ant vs Maven?.] Maven on käytössä Metropolian Ohjelmistotuotantoprojekti-kurssin ohjelmistoprojektien koontityökaluna.

Apache Maven on ohjelmistoprojektin hallintatyökalu. Se perustuu projekti-olio-malli -konseptiin (Project Object Model, POM). Maven kykenee kokoamaan projektin, testaa-

maan projektin, antamaan käyttäjälle palautetta, dokumentoimaan kokoamisen ja tekemään useita muita tehtäviä erillisillä Mavenin tarjoamilla lisäosilla (plug-ins). [Apache Maven Project: Welcome to Apache Maven.]

POM on XML-tiedosto, joka sisältää informaatiota projektista ja yksityiskohtaisen konfiguraation, jota Maven käyttää projektin kokoamiseen. POM kostuu kohteista (goals), jotka on määritelty *pom.xml* -tiedostossa. Kun suoritetaan erillinen tehtävä tai kohde, Maven etsii POM:mia työhakemistosta, lukee POM:min, hakee tarvittavan konfiguraation ja suorittaa kohteen.

Eräitä POM:issa määriteltäviä konfiguraatioita ovat projektin riippuvuudet, käytettävät lisäosat, ajettavat kohteet ja koontiprofiilit. POM:iin voi määritellä muutakin tietoa, kuten projektin version, kehittäjät ja postituslistat. [Apache Maven Project: Introduction to the POM.]

Maven on Java-työkalu, joten Java täytyy olla asennettuna Mavenia ajavalle koneelle. Tämä luku perustuu Apachen verkkosivuilla sijaitsevaan Mavenin käyttöohjeeseen. [Apache Maven Project: Maven in 5 Minutes.] Seuraavaksi luetellaan eri vaiheet Maven-projektin luomiseksi.

Ensin on ladattava Maven ja käytävä läpi asennusohjeet, jotka löytyvät osoitteesta [http://maven.apache.org/download.html#Installation]. Tämän jälkeen on avattava terminaali tai komentokehote ja kirjoitettava:

```
mvn --version
```

Tämän tulostaa asennetun Mavenin ja Javan versio. Esimerkiksi:

```
1. Apache Maven 3.0.3 (r1075438; 2011-02-28 18:31:09+0100)
2. Maven home: D:\apache-maven-3.0.3\bin\..
3. Java version: 1.6.0_25, vendor: Sun Microsystems Inc.
4. Java home: E:\Program Files\Java\jdk1.6.0_25\jre
5. Default locale: nl_NL, platform encoding: Cp1252
6. OS name: "windows 7", version: "6.1", arch: "amd64", family: "windows"
```

Maven-projekti luodaan ajamalla terminaalissa Maven kohde (goal):

```
mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app -
DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

Ensimmäisellä ajolla tämä komento lataa tarvittavat Maven lisäosat ja muita tiedostoja paikalliseen tietovarastoon. Huomataan, että ”generate” kohde loi ”artifactId”:nä annetun hakemiston.

Kirjoitetaan komentoriville:

```
cd my-app
```

Tämän kansion alta löytyy Mavenin standardi projektirakenne:

```
my-app
|-- pom.xml
`-- src
    |-- main
    |   `-- java
    |       `-- com
    |           `-- mycompany
    |               `-- app
    |                   `-- App.java
    `-- test
        `-- java
            `-- com
                `-- mycompany
                    `-- app
                        `-- AppTest.java
```

Yllä kuvatussa hakemistorakenteessa oleva ”src/main/java”-hakemisto sisältää projektin lähdekoodin. Vastaavasti ”src/test/java”-hakemisto sisältää testien lähdekoodin. *Pom.xml*-tiedosto kuvaa projektin konfiguraatiota ja sisältää projektin kokoamisohjeet. Esimerkkiprojektin POM näyttää tältä:

```
1. <pro-
   ject xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/20
   01/XMLSchema-instance"
2.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org
   /xsd/maven-4.0.0.xsd">
3.   <modelVersion>4.0.0</modelVersion>
4.
5.   <groupId>com.mycompany.app</groupId>
6.   <artifactId>my-app</artifactId>
```

```
7.    <version>1.0-SNAPSHOT</version>
8.    <packaging>jar</packaging>
9.
10.   <name>Maven Quick Start Archetype</name>
11.   <url>http://maven.apache.org</url>
12.
13.   <dependencies>
14.     <dependency>
15.       <groupId>junit</groupId>
16.       <artifactId>junit</artifactId>
17.       <version>4.8.2</version>
18.       <scope>test</scope>
19.     </dependency>
20.   </dependencies>
21. </project>
```

POM-tiedosto on Maven-projektin konfiguraation ydin. Se sisältää suurimman osan informaatiosta, joka vaaditaan projektin kokoamiseen halutulla tavalla.

Aikaisemmassa esimerkissä ajettiin Mavenin kohde (goal) *archetype:generate,* ja annettiin sille parametreja. Etuliite *archetype* on lisäosa, joka sisältää kohteen *generate*. Tätä voi verrata Antin tehtävään. Tämä kohde loi yksinkertaisen projektin, joka perustuu arkkityyppiin. Yleisesti Mavenin lisäosat koostuvat kohteista, joilla on yhteinen päätarkoitus. Esimerkki tällaisesta lisäosasta on *jboss-maven-plugin,* jonka tehtävänä on käsitellä erinäisiä Jboss-komponentteja (items).

Maven-projekti kootaan komentoriviltä seuraavanlaisesti:

```
mvn package
```

Komentorivi tulostaa eri komentoja, tulostus loppuu seuraavanlaisesti:

```
...
[INFO] ------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------
[INFO] Total time: 2 seconds
[INFO] Finished at: Thu Jul 07 21:34:52 CEST 2011
[INFO] Final Memory: 3M/6M
[INFO] ------------------------
```

Toisin kuin ensimmäiseksi suoritetussa komennossa (*archetype:generate*) huomataan, tällä kertaa käytetty komento *package* on vain yksi sana. Tällaista komentoa kutsutaan

kohteen (goal) sijaan vaiheeksi (phase). Vaihe on yksi askel koonnin elinkaaressa (build lifecycle), joka on järjestetty sarja eri vaiheita. Jos esimerkiksi ajetaan vaiheen *compile*, Maven itseasiassa suorittaa sarjan vaiheita:

1. validate
2. generate-sources
3. process-sources
4. generate-resources
5. process-resources
6. compile

Compilen ajamisen jälkeen voidaan testata käännettyä JAR:ia seuraavalla komennolla:

```
java -cp target/my-app-1.0-SNAPSHOT.jar com.mycompany.app.App
```

Tämä komento tulostaa:

```
Hello World!
```

Käydään läpi vielä eri Mavenin yleisimmät oletuselinkaaressa käytetyt vaiheet. Näitä ovat:

-validate: varmistaa, että projekti on ehjä ja kaikki tarvittava tieto on saatavilla.
-compile: kääntää projektin lähdekoodin.
-test: testaa käännetyn lähdekoodin käyttäen sopivaa yksikkötestikehystä. Näiden testien ei tulisi vaatia vaiheita *package* tai *deploy.*
-package: paketoi käännetyn koodin sen jaettavaan muotoon, esimerkiksi JAR:riksi.
-integration-test: prosessoi ja järjestää paketin ympäristöön, jossa integraatiotestejä voidaan ajaa, jos tarpeellista.
-verify: tarkistaa onko paketti validi ja täyttääkö se laatukriteerit.
-install: asentaa paketin paikalliseen tietovarastoon, muiden paikallisten projektien käyttöön tai paikallisten projektien riippuvuudeksi.
-deploy: kopioi lopullisen paketin ulkoiseen tietovarastoon jaettavaksi muille kehittäjille ja projekteille. Tehdään integraatio- tai julkaisuympäristössä.

On vielä kaksi muuta mainitsemisen arvoista Maven-elinkaarta, jotka eivät sisälly yllä listattuun oletuselinkaareen. Näitä ovat:

-clean: poistaa aikaisempien koontien yhteydessä luodut artifaktit.
-site: generoi verkkodokumentaation kyseiselle projektille

Jokainen vaihe sisältää sarjan suoritettavia kohteita, joiden ajaminen riippuu projektin paketointityypistä. Jos projektin tyyppi on JAR, niin *package* ajaa kohteen *jar:jar.* Vastaavasti WAR -tyyppisen projektin *package* -vaihe suorittaa kohteen *war:war.*

3.4 Koonti ja käyttöönotto

Ohjelmiston saattamista lähdekoodista loppukäyttäjälle käyttökelpoiseen muotoon kutsutaan koonniksi. Lähdekoodin kääntämisen lisäksi koonti voi sisältää monia muita vaiheita. Aleksi Lukkarinen on opinnäytetyössään listannut esimerkkejä erilaisista koonneista seuraavasti: [Aleksi Lukkarinen, opinnäytetyö].

-ohjelmointikielisen lähdekoodin sekä resurssien automatisoitua luomista
-eri ohjelmointikielillä tuotetun lähdekoodin kääntämistä kone- ja/tai välikielille
-väli- tai konekielisen ohjelmakoodin jälkikäsittelyä esimerkiksi testauksen, aspektiohjelmoinnin tai koodin tarkoituksellisen sotkemisen vuoksi
-sovelluksen testausta (esimerkiksi toimivuus, tietoturvallisuus, suorituskyky, rakenteellinen ja koodaustekninen laatu sekä käyttötarkoitustaan vastaavuus)
-dokumentaation koontia (kuten varsinaisen sovelluksen koonti)
-koontituotteiden paketointia sekä mahdollisten fyysisten jakelumedioiden luontia toimivuus- ja virustarkistuksineen
-ohjelmistomoduulien integrointia
-palautteen antamista koonnin tuloksista.

Lukkarinen jäsentelee opinnäytetyössään sovelluksen jakeluun ja käyttöönottoon sisältyviksi asioiksi:

-Sovellusta suorittavien ja tukevien palvelinympäristöjen luominen ja päivittäminen.
-Sovelluksen sekä sen käyttämien ohjelmistokomponenttien siirtäminen ja määrittäminen esimerkiksi testaus-, laadunvarmistus-, esittely-/koulutus- sekä tuotantoympäristöihin. Kohdeympäristöissä olevat vanhat versiot on mahdollisesti päivitettävä, jollei uusia versiota asenneta niiden rinnalle.

-Uusien tietokantojen luominen sekä vanhojen päivittäminen uutta sovellusversiota vastaaviksi.
-Kirjanpidollisen materiaalin tuottaminen ja ylläpitäminen.
-Palautteen antaminen jakelu- ja käyttöönottoprosessin onnistumisesta.

Tämän luvun tarkoitus oli perehdyttää lukija ymmärtämään koontiprosessia koontikuvaus-esimerkkien avulla. Koontiprosessin ymmärtäminen on välttämätöntä seuraavan luvun ymmärtämiseen, jossa perehdytään koontiympäristö -käsitteeseen.

4 Koontiympäristön tarkkailun kriittiset sovellukset Jenkins CI sekä Agilefant

Tässä luvussa luodaan nopea katsaus koonti- ja käyttöönottoprosessien automatisoimiseen, tutustutaan jatkuvaan integraatioon sekä sovellukseen Jenkins CI.

Lisäksi tutustutaan projektihallintaohjelmaan Agilefant. Agilefant on yksi monipuolisimmista kevyistä vapaan lähdekoodin projektinhallintasovelluksista, ja se soveltuu niin pieniin kuin suuriinkin projekteihin.

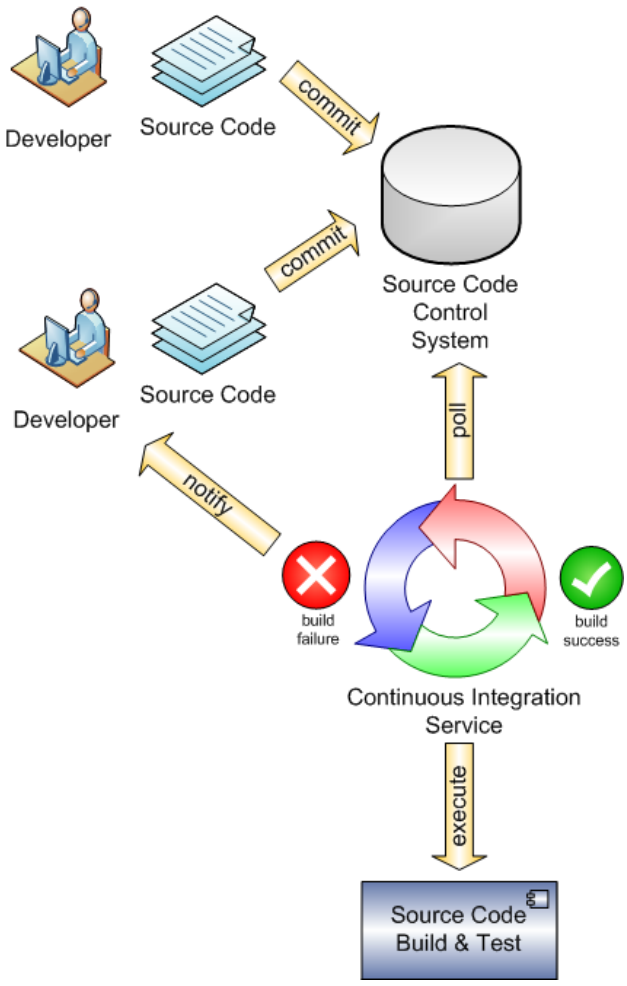
Nämä sovellukset esitellään pohjustuksena lukuja 5 ja 6 varten, joissa tutustutaan koontiympäristön tarkkailuun.

4.1 Jatkuva integraatio

Jatkuva integraatio (Continuous Integration, CI), on modernin sovelluskehityksen kulmakivi. Itse asiassa se on mullistanut sovelluskehityksen. Kun jatkuva integraatio otetaan käyttöön organisaatiossa, se muuttaa radikaalisti sitä, miten työryhmä ymmärtää koko kehitysprosessin. Jatkuvalla integraatiolla on potentiaalia laukaista sarja parannuksia organisaation tuotteen kehitysprosessiin. Se alkaa yksinkertaisesta automaattisesta koonnista kehittyen systeemiksi, joka automaattisesti saattaa tuotteen tuotantoon. Hyvä CI-infrastruktuuri mahdollistaa virheiden helpon sekä nopean huomaamisen ja tarjoaa kojelaudan projektin hyvinvoinnin ja muiden statistiikkojen seuraamiseen sekä kehittäjille, että muille projektin osapuolille. Jenkins The Definite Guiden mukaan

jokaisen kehitysryhmän, kuinka pienen tahansa, tulisi harjoittaa jatkuvaa integraatiota. [Jenkins The Definitive Guide 2011: s. 1.]

Jatkuva integraatio -palvelinsovelluksen ympärille rakennettu kehitysympäristöinfrastruktuuri koostuu yksinkertaisimmillaan kehittäjän IDE-kehitystyökalusta, versionhallinnasta, koontityökalusta, näiden sovellusten konfiguraatiosta ja siitä kuinka nämä osapuolet kommunikoivat keskenään. Jatkuva integraatio on lyhyesti sanottuna myös lähdekoodin, dokumentaation ja resurssien jatkuvaa yhteen kasaamista, projektille yhteiseen paikkaan. Kuvassa 7 esitetään jatkuvan integraation prosessi.



Kuva 7. Jatkuva integraatio [Continuous Integration – Basic Overview and Best Practices]

Kuvassa on esitelty jatkuva integraatio prosessi, jonka osapuolina toimivat kehitysryhmä, versionhallinta sekä jatkuva integraatio palvelu (CI-palvelu, esim. Jenkins CI). CI-palvelu tarkkailee versionhallintaa. Kun CI-palvelu huomaa muutoksen versionhallin-

nassa, se kokoaa versionhallinnassa olevan sovelluksen ja ajaa sovellukseen liitetyt testit. CI-palvelu huomauttaa kehittäjiä koonnin epäonnistuessa.

Ohjelmistoprojektien kehitysryhmät koostuvat usein useista kehittäjistä, nämä kehittäjät tuottavat esimerkiksi lähdekoodia sekä dokumentaatiota. Jos projektissa ei ole otettu käyttöön jatkuvaa integraatiota, on projektin kokoaminen tehtävä säännöllisin väliajoin manuaalisesti, joka on epäkäytännöllistä ja vie aikaa. Tätä varten on kehitetty jatkuva integraatio -palvelinsovelluksia (luku 4.2), joiden tehtävänä on valvoa, että koonti pysyy ehjänä jokaisen kehittäjän tekemän tietovarastopäivityksen (integraatio) tapahtuessa.

Hyvin suunniteltu ja toteutettu kehitysympäristöinfrastruktuuri mahdollistaa esimerkiksi seuraavanlaisen automatisoidun sarjan toimintoja: Ohjelmoija muuttaa tietovarastossa sijaitsevaa tiedostoa, jonka jälkeen integraatiopalvelinsovellus (esimerkiksi Jenkins CI) reagoi automaattisesti tietovarastopäivitykseen ja käynnistää ennalta määrätyn sarjan toimintoja ennalta määrätyssä järjestyksessä. Lopputuloksena voi olla esimerkiksi loppukäyttäjälle valmis yksikkötestattu tuote automaattisesti generoidulla dokumentoinnilla tai rikkinäinen koonti. Hyvin hallitussa projektissa tämä tapahtuu yhdellä napin painalluksella.

Rikkinäinen koonti tarkoittaa esimerkiksi sitä, että ohjelmistomoduulit tai -komponentit ovat konfliktissa keskenään. Rikkinäinen koonti voi tarkoittaa myös yksittäisen yksikkötestin epäonnistumista tai jopa xml-tiedoston sisennyksien poikkeamista ennalta sovitusta linjasta riippuen integraatiopalvelinsovelluksen konfiguraatioista.

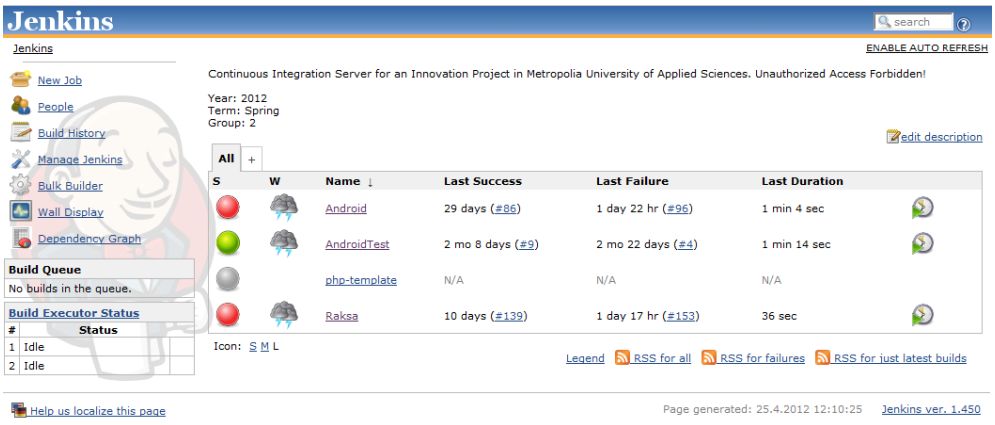
4.2 Jenkins CI

Jenkins CI on jatkuva integraation -palvelinsovellus, joka tarkkailee ja auttaa käyttäjää hallinnoimaan sekä luomaan koontitehtäviä tai projektin aikana useasti suoritettavia, toistuvia tehtäviä.

Koontitehtävä tarkoittaa yksittäistä tehtävää, joka suoritetaan yhdessä muiden koontitehtävien kanssa, kun projekti kootaan (build). Koontitehtävä voi olla esimerkiksi ohjelmiston yksikkötestaaminen. Projektin aikana useasti suoritettava koontitehtävä voi olla esimerkiksi sähköpostimuistutus viikkopalaverista.

Jenkins CI tarjoaa käyttäjäystävällisen käyttöliittymän projektin kokoamisen hallintaan sekä työkalut myös ulkoisten koontitöiden (build jobs) tarkkailuun.

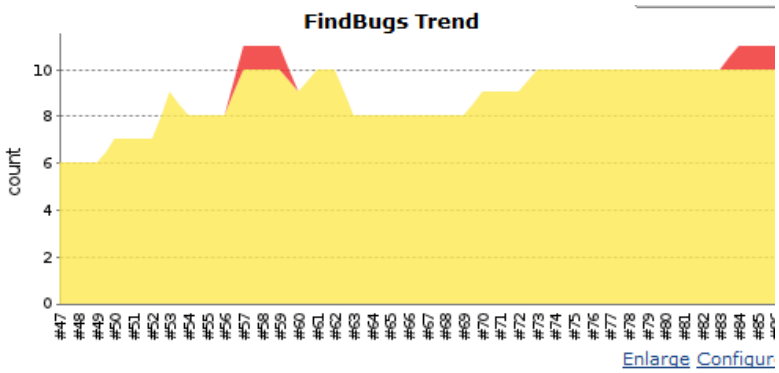
Kuvassa 8 on esitelty Jenkins CI:n päänäkymä. Esimerkkiprojektiin on määritelty neljä koontityötä: *Android, AndroidTest, php-template* ja *Raksa*. Näihin töihin on liitetty erilaisia koontitehtäviä. Kun näistä töistä kootaan yksi, käynnistää Jenkins CI kaikki yksittäiset, tähän työhön liitetyt koontitehtävät. Näiden koontitehtävien välille voidaan asettaa riippuvuuksia. Koontitehtävät voivat olla riippuvaisia myös ulkopuolisista koontitöistä tai projekteista.



Kuva 8. Jenkins CI:n päänäkymä

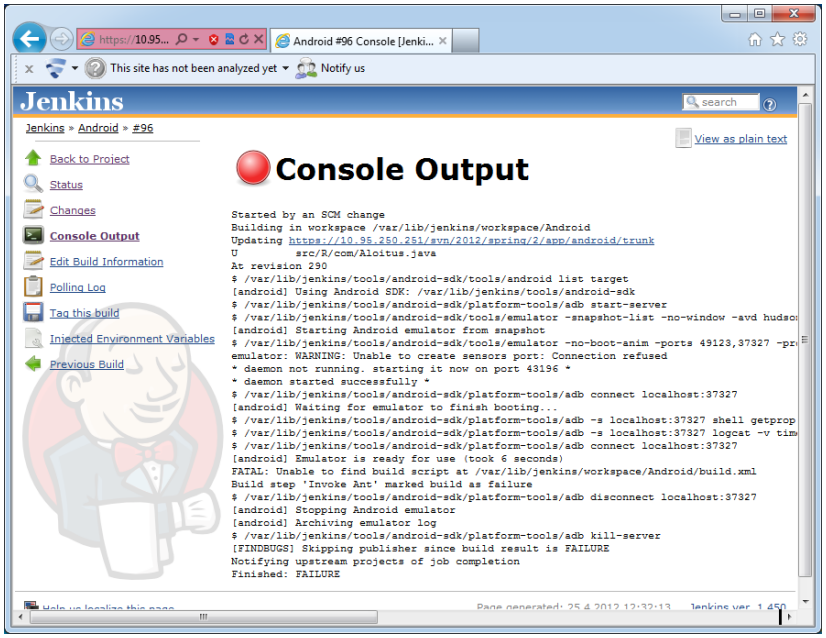
Koontitehtävien (luku 3) ja koontitehtävien riippuvuuksien älykäs määrittely mahdollistaa dynaamisen koontiprosessin tekemisen. Dynaamisella koontiprosessilla tarkoitetaan sitä, että projektin kokoaminen ei ole aina samanlainen prosessi, vaan on riippuvainen esimerkiksi kellonajasta tai toisesta koontityöstä.

Jenkins CI tarjoaa paljon erilaisia lisäosia (plug-in) projektin kokoamiseen ja tarkkailuun. Jenkinsin lisäosakirjastosta on ladattavissa esimerkiksi koodikattavuuslisäosa, testituloslisäosa, erilaisia koonnin laukaisulisäosia ja satoja muita lisäosia. Lisäosien lataaminen ja asennus käy helposti Jenkinsin omasta käyttöliittymästä. Kuvassa 9 on esitelty erään Jenkins CI -lisäosan tuottama kuvaaja. Tästä kuvaajasta on luettavissa ohjelman lähdekoodista löytyvien virheiden määrä koontien #47 ja #86 välillä.



Kuva 9. FindBugs Trend

Koontitehtävien tarkkailu ei rajoitu kuitenkaan pelkästään kuvaajiin. Jenkins CI antaa myös yksityiskohtaista tietoa koonnista ja sen eri vaiheista (kuva 10).



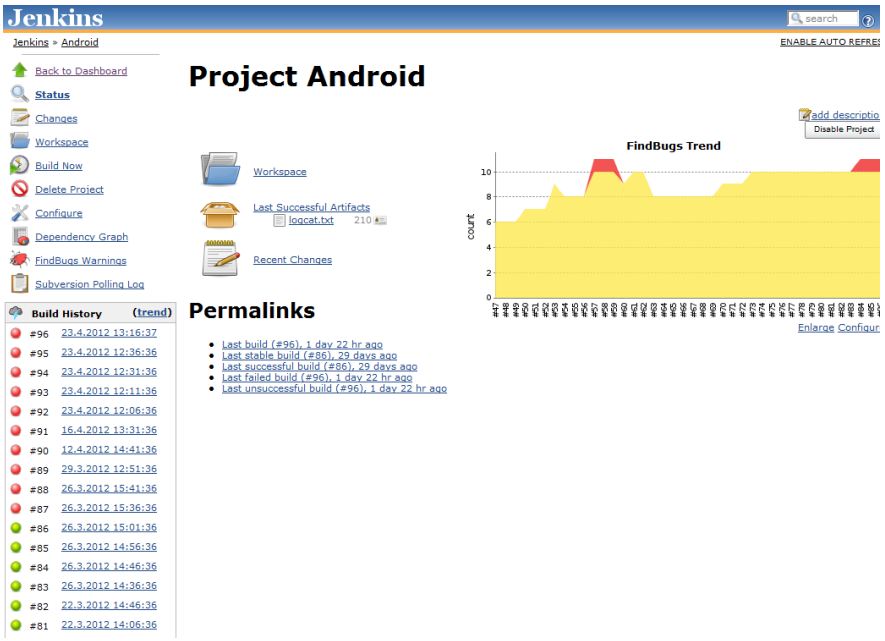
Kuva 10. Jenkinsin konsolinäkymä

Käyttäjä pääsee myös katsomaan aikaisempien koontien kuvaajia ja koontilokeja (kuva 11) valitsemalla aikaisemman koonnin koontihistoria. Koontihistoria sijaitsee koontityön projekti-näkymän vasemmassa reunassa (kuva 12). Vihreä ympyrä kuvan vasemmassa reunassa tarkoittaa onnistunutta koontia. Punainen ympyrä tarkoittaa, että koonti ei ole mennyt läpi.

Kuva 11. Jenkins CI:n koontihistoria

Jokaisella työllä on erillinen päänäkymä Jenkins CI:ssä (kuva 12). Tässä näkymässä näkyy osa kuvaajista ja käyristä, joita Jenkins CI koonnin yhteydessä tuottaa.

Kuvassa 11 on esitelty *Android* koontityön projekti-näkymä. Tästä kuvasta on nähtävissä, että koontityössä *Android* on otettu käyttöön *FindBugs*-lisäosa. Tämä lisäosa on tuottanut projektin päänäkymään kuvaajan, josta on luettavissa ohjelmiston lähdekoodissa olevien virheiden määrä koontien #47 ja #86 välillä.



Kuva 12. Jenkins CI:n koontityön päänäkymä

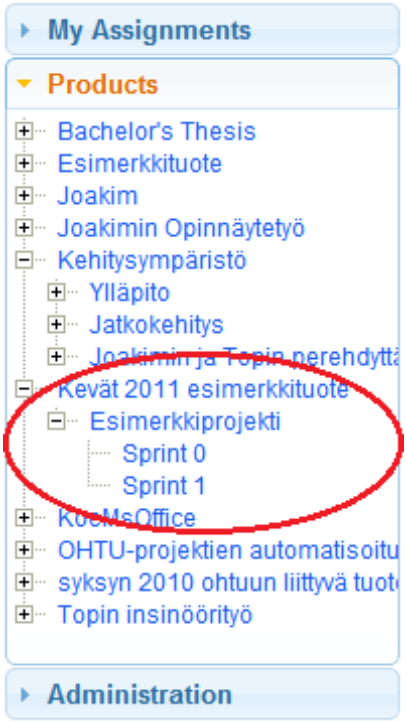
Esimerkkikuvassa olevat kuvaajat ovat oleellisia opinnäytetyön kannalta. Nimittäin myöhemmin (luku 6) käsiteltävä opinnäytetyön konkreettinen osa, ohjelmistoprojektin tarkkailuun käytettävä sovellus: ”Koonta” käyttää näitä kuvaajia hyväkseen välittäessään informaation koonnin tilasta Ohjelmistoprojekti-kurssin opiskelijoista koostuvalle kehitysryhmälle. Nämä kuvaajat näkyvät luvussa 2 esitellyllä näytöllä, joka on tarkoitettu vain koontiympäristön seurantaan.

4.3 Agilefant

Agilefant on tällä hetkellä yksi yksinkertaisimmista ketterien menetelmien hallintatyökaluista. Se tuo yhteen sekä suuren että pienen mittakaavan ajankäyttösuunnittelun. Agilefant sopii työryhmälle, jonka on pystyttävä suunnittelemaan tulevia työtunteja sekä hallinnoimaan jo kerääntynyttä työkertymää. Agilefant toimii myös suuren mittakaavan projektin hallinnointisovelluksena usealla työryhmällä. [Agilefant: what is Agilefant?]

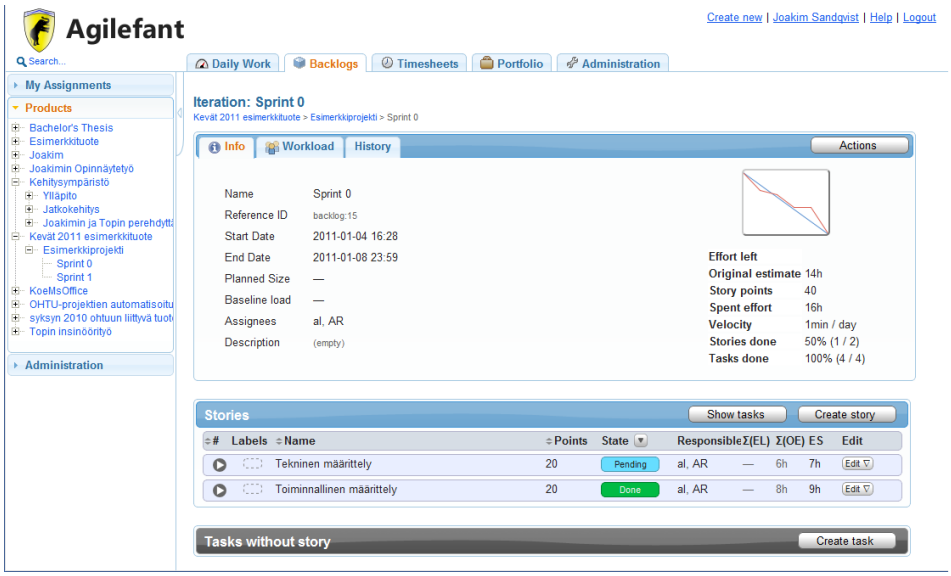
Kun Agilefant otetaan projektiryhmän käyttöön, on Agilefantiin luotava ensin tuote. Yksi Agilefant -asennus voi sisältää monia tuotteita (Product). Tuotteeseen voidaan liittää monta projektia (Project). Projektiin voidaan liittää monta iteraatiota (Iteration) ja iteraatioihin tarinoita (Story). Tarinoihin puolestaan liitetään tehtäviä (Tasks). Tehtäviin määritellään arvioitu työaika, tehty työaika ja jäljellä oleva työaika. Ennen tätä sovellukseen voidaan halutessa lisätä käyttäjiä, mutta ohjelma ei vaadi tätä.

Kuvassa 13 on esitelty esimerkki tuotteiden puurakenteesta. Kuvassa ”Kevät 2011 esimerkkituote” on ”Kevät 2011 esimerkkituote” -niminen tuote. Tähän tuotteeseen on liitetty yksi projekti: ”Esimerkkiprojekti”. Tähän projektiin on liitetty iteraatioita: ”Sprint 0” ja ”Sprint 1”.



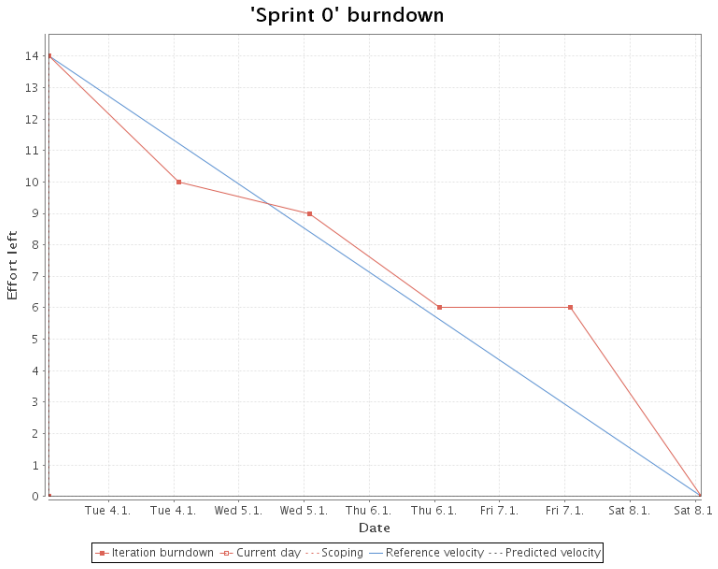
Kuva 13. Esimerkki Agilefantin tuotteiden puurakenteesta

Kuvassa 14 on esitelty Agilefantin iteraationäkymä. Iteraatioon on määritelty tarinat (stories): ”Tekninen määrittely” ja ”Toiminnallinen määrittely”. Tarinoihin on liitetty vastuuhenkilöt: ”al” ja ”AR”. Tarinoiden tilat ovat: ”Pending” ja ”Done”. Tekniseen määrittelyyn käytetty aika (ES, Effort Spend) on 7 tuntia. Alkuperäinen arvio (OE, Original Estimate) on määritelty 6 tuntiin ja jäljellä oleva työaika (EL, Effort Left) ei ole määritelty ollenkaan. Agilefant osaa generoida tästä informaatiosta ”Burndown Chartin”. Burndown Chart antaa osviittaa siitä, onko projektin iteraatio aikataulussa.



Kuva 14. Agilefantin iteraatio -näkymä

Kuvassa 15 oleva sininen käyrä kuvaa iteraation optimaalista ajankäyttöä. Punainen käyrä kuvaa toteutunutta ajankäyttöä. Iteraatioon liitettyjen tehtävien valmistuessa ajallaantulisi punaisen käyrän liittyä sinineen käyrään kuvaajan oikeassa alakulmassa.



Kuva 15. Agilefantin Burndown Chart

Kuvassa 16 oleva kuvaaja on oleellisessa osassa opinnäytetyössä. Johdannossa ensimmäisen kerran mainittu ja myöhemmin (luku 6) esiteltävän sovelluksen Koonta on tarkoitus käyttää myös tätä kuvaajaa hyväkseen välittäessään informaatiota projektin tilasta kehitysryhmälle, mutta tätä toiminnallisuutta ei ole vielä implementoitu.

5 Koontiympäristön tarkkailu

Laadukkaan koodin kirjoittaminen jokaiseen julkaisuun vaatii paljon huomiota. Projektin etenemisen kannalta on ekonomista, jos ei tarvitse koko ajan katsoa sivupeilistä, menikö kaikki hyvin. On parempi, että kehittäjälle ilmoitetaan aktiivisesti, kun jokin asia vaatii huomiota. [Pragmatic Project Automation 2004: s. 125.]

5.1 Yleistä koontiympäristön tarkkailusta

Tässä luvussa perehdytään koontiympäristön tarkkailuun. Selitetään ensin, mikä tekee koontiympäristön tarkkailemisesta tarpeellista ja mitä eri lähestymistapoja koontiympäristön tarkkailemiseen voidaan ottaa.

Mahdollisimman nopea ilmoitus rikkoutuneesta koonnista on tärkeää. Ilmoitus koonnin rikkoutumisesta tarkoittaa sitä, että muutosten tekeminen täytyy lopettaa, kunnes koonti on korjattu. Jos koontia ei korjata, saattaa tehty työ olla turhaa. Jos rikkinäiseen koontiin pystytään reagoimaan nopeasti, on virheen jäljittäminen helpompaa, sillä muutosten määrä ehjän ja rikkinäisen koonnin välillä on minimaalinen. Koonnin korjaaminen heti on myös helpompaa kuin myöhemmin, kun ongelmat ovat kasaantuneet päällekkäin.

Yksi hyvä tapa ilmoittaa koonnin tilasta on tehdä tämä sähköpostitse, mutta mitä jos sähköpostia ei ole lähettyvillä. Koonnin tilasta voi ilmoittaa tekstiviestillä. Ilmoittaminen on mahdollista myös tehdä RSS-syötteellä. Kehittäjä voi tilata ja lukea tämän tiedoston haluamallaan RSS-lukijalla. [Pragmatic Project Automation 2004: s. 125.] Määrittelemällä tietynlainen koontitehtävä esimerkiksi Mavenin *pom.xml* (luku 3.3) -tiedostoon, voidaan saada koonnin tila näkymään vaikka verkkosivulla.

Nykypäivänä ei ole harvinaista, että ohjelmistokehitysryhmät sijaitsevat eri puolilla maapalloa. Esimerkkinä käytetään kahta kuviteltua kehitysryhmää. Toinen ryhmä on Intiasta ja toinen Amerikasta. Ryhmien työvuorojen aikaero on 12 tuntia. Toinen ryhmä aloittaa ohjelmoinnin ja toinen ryhmä menee kotiin. Toisen ryhmän saapuessa vuoroonsa on koonnin oltava vakaassa tilassa. Tämä takaa sen, että toisen ryhmän on helppo jatkaa siitä, mihin toinen jäi. [Pragmatic Project Automation 2004: s. 125.]

Ohjelmistokehityksessä on kyse yhtälailla kommunikaatiosta kuin lähdekoodin kirjoittamisesta. Kun ryhmät ovat eri paikoissa, kommunikaatio kärsii, kun tiedon välittäminen eteenpäin kahvipöydässä ei enää onnistukaan. Täten toisistaan erossa olevien ryhmien on otettava käyttöön jokin menetelmä, jolla he saavat tiedon siitä, että koonti on onnistunut. Tämä antaa kummallekin ryhmälle luottamusta jatkaa eteenpäin pelkäämättä sitä, että he rakentaisivat uutta koodia rikkinäisen koonnin päälle ja lisäisivät töitä entisestään.

Informaation koonnin tilasta voi siis saada RSS-syötteenä, sähköpostina, tekstiviestinä tai verkkosivulta. Informaation saaminen tähän tapaan on kuitenkin vanhanaikaista. Kun CI-infrastruktuurin on rakenteellisesti selkeä, mahdollistaa se XF-laitteiden (välittömän palautteen laite) helpon implementaation ja koonnin tila saadaan välitettyä tehokkaammin ja visuaalisemmin kuin perinteisillä menetelmillä. XF-laitteiden käyttöönotto ei suinkaan syrjäytä perinteisiä menetelmiä, vaan vahvistaa projektin hallintaa ja ohjelmistoprojektin tilan tarkkailua entisestään.

Kuten yllä mainittiin, XF-laitteet eivät syrjäytä perinteisiä menetelmiä ilmoittaa koonnin tilasta. Selkein ero perinteisten ja modernien menetelmien välillä on se, että perinteiset menetelmät soveltuvat paremmin etätyöskentelyyn ja XF-laitteet soveltuvat paremmin toimistotyöskentelyyn, kun informaatio halutaan välittää kehittäjille välittömästi. XF-laitteista kerrotaan lisää seuraavassa luvussa.

Ilmoittaminen koonnin epäonnistumisesta on olennainen osa CI-strategiaa. Esimerkkejä perinteisistä ratkaisuista olivat aiemmin mainitut: sähköposti, RSS-syöte, verkkosivu sekä tekstiviesti. Kaikki ilmoittamiseen käytetyt ratkaisut eivät kuitenkaan sovi kaikkiin työympäristöihin ja -kulttuureihin.

5.2 Välittömän palautteen laitteet

Eräs klassinen esimerkki XF-laitteesta (Xtreme Feedback Device) on laavalamppu. Perusidea on, että koonnin epäonnistuessa laavalamppu on punainen ja koonnin onnistuessa laavalamppu on vihreä. Toinen esimerkki koonnin epäonnistumisesta ilmoittavasta laitteesta on Retaliation Projektin [Who Broke The Build? 2011] tuotos: vaahtomuoviraketteja ampuva ohjuspatteri. Koonnin epäonnistuessa patteri ampuu koonnin rikko-

nutta ohjelmoijaa. Näitä laitteita kutsutaan englanniksi nimellä ”extreme notification devices” tai vaihtoehtoisesti ”extreme feedback devices”. Vuosia Apache Software Foundation -projektissa mukana ollut Carlson Sanchez leikillään ilmoittaa ”Software at the end of the universe” -blogissaan lempilaitteensa olevan sähkötuoli. [Using Lava Lamps for Continuous Integration 2009.] Tieto koonnin tilasta voidaan siis välittää käyttäjälle monilla erilaisilla laitteilla.

Olipa työympäristössä käytetty koonnin tilaa ilmaiseva laite (suoran palautteen laite) sitten liikennevalo, puhuva jänis tai jokin vielä eksoottisempi ratkaisu, on selvää, että tällaisilla ratkaisuilla voi olla tässä luvussa mainittujen konkreettisten hyötyjen lisäksi myös työilmapiiriä ja -moraalia nostattavia vaikutuksia. [Jenkins The Definitive Guide 2011: s. 221.]

Tässä luvussa tutustutaan Saksassa sijaitsevan Softwareschneiderei yrityksen XF-laite-ratkaisuihin, joilla pyritään ratkaisemaan erilaisia projektin hallintaan, jatkuvaan integraatioon, sekä extreme feedback -laitteisiin liittyviä ongelmia. Yrityksen blogissa (Schneide Blog) verkkojulkaisuissa esitellään yleensä ohjelmistoprojektiin liittyvä ongelma, sen mahdollinen ratkaisu ja ratkaisun seuraukset. Luvuissa 5.2.1 ja 5.2.2 tutustutaan kahteen mielenkiintoiseen blogissa esiteltyyn ratkaisuun: The Code Flow-O-Meter ja Smell-O-Mat.

5.2.1 The Code Flow-O-Meter

Yhtenä ketterän tai hyvän ohjelmoinnin lähtökohtana on tehdä tietovarastopäivityksiä aikaisin ja usein. Ongelma, jonka Softwareschneiderei kohtasi, oli seuraavanlainen. Jokaisen pienen tietovarastopäivityksen tapahtuessa muutokset käyvät läpi erilaisia testejä. Testien ja laaduntarkkailun lisääntyessä ohjelmoijat halusivat pitää oman versionsa paikallisena mahdollisimman pitkään, jotta se pääsisi läpi testeistä. Tämä hidastaa ohjelmistokehitystä. Varsinkin aloittelevampien kehittäjien kesken ”Se on valmis sitten, kun se on tehty” -lauseella oikeutettiin väljät tietovaraston päivitysintervallit. Tämä on kuitenkin paras tapa olla huomaamatta aikaista palautetta - ja aikainen palaute on toimivaa palautetta. Softwareschneidereillä päätettiin asentaa liikutettava suihkulähde (kuva 16) vastaiskuksi myöhäisiä tietovarastopäivityksiä vastaan.

Kuva 16. The Code Flow-O-Meter

Softwareschneidereillä ohjelmoitiin pieni IRC-botin, joka laukaisee suihkulähteeseen kiinnitetyn sovellusmoduulin. Suihkulähdettä pystyttiin hallinnoimaan nyt IRC:n kautta. Tämän jälkeen tehtiin pieni skripti, joka käsitteli tietovarastopäivityksistä saadut viestit ja määritti päivitykselle painoarvon perustuen muutettujen tiedostojen määrään. Päivityksen koko ei lisännyt yhtä paljon painoarvoa kuin päivitystapahtuma itse. Painoarvo määräsi veden virtaamisajan suihkulähteessä. Pelin nimi oli selvä: pidä virtaus käynnissä. Blogin mukaan tulokset olivat selvästi nähtävissä. Blogissa sanotaan laitteen toimineen ja kaikkien pitävän laitteesta. Blogissa ei kuitenkaan mainita, tiheníkö päivitysintervalli.

5.2.2 Smell-O-Mat

Toinen Schneide Blogissa esitelty kokeellinen XF-laite, Smell-O-Mat (kuva 17), antaa visuaalisen palautteen sijaan hajuaistilla aistittavan palautteen. [Schneide Blog 2009: Smell if its well.]

Kuva 17. Smell-O-Mat

Tekninen ratkaisu Smell-O-Matissa on sama kuin ylempänä esitellyssä XFD:ssä. Smell-O-Mat käyttää IRC-bottia välikappaleena tiedon siirtämiseen palvelimen ja XFD:n välillä. Smell-O-Mat käyttää kahta eri koodihajua: hyvää ja pahaa hajua. Hyväksi tuoksuksi valittiin greipin tuoksu. Tekstissä ei mainita, mikä valittiin pahaksi hajuksi. Softwareschneideredei ei käytä usein tätä XFD:tä, mutta aina välillä joku laukaisee laitteen. Kun toimistossa tulee tarvetta raikkaalle tuoksulle, käy joku korjaamassa vanhan koodinpätkän.

Vaikka tämä ratkaisu on melko epäkäytännöllinen, on se kuitenkin hauska. Näiden laitteiden lisäksi yrityksellä on käytössä - tai on ollut käytössä XF-laitteita, jotka ovat aistittavissa melkein kaikilla ihmisen aisteilla: tunto-, näkö-, kuulo- sekä hajuaisteilla. Vain makuaistia hyödyntävä laite on toteuttamatta.

6 Koontiympäristöntarkkailu sovellus

Osana opinnäytetyötä oli tehtävänä ohjelmoida Javalla ohjelma, joka tarkkailee koontiympäristöä. Lähestymistapa koontiympäristön tarkkailuun on erilainen kuin luvussa 5 mainituissa menetelmissä. Koonta antaa käyttäjälle mahdollisuuden tuoda kuvia tai

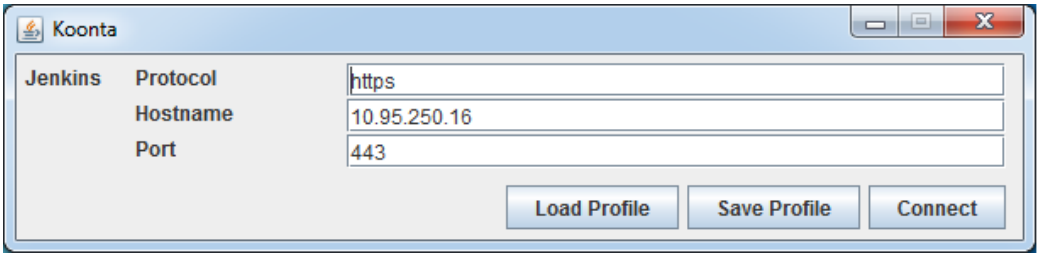
kuvaajia ohjelman graafiseen näkymään. Tässä tapauksessa kuvat ovat esimerkiksi Jenkins CI -lisäosien tuottamia kaavioita tai Agilefantin tuottamia burndown chartteja. Sovelluksen tarkoitus on tehdä kehitysryhmän työympäristöstä informatiivisempi.

Seuraavassa luvussa perehdytään sovelluksen toiminnallisuuteen. Tämän jälkeen tutustutaan sovelluksen kehittämisprosessiin, koodiin ja arkkitehtuuriin.

6.1 Sovelluksen toiminnallisuus

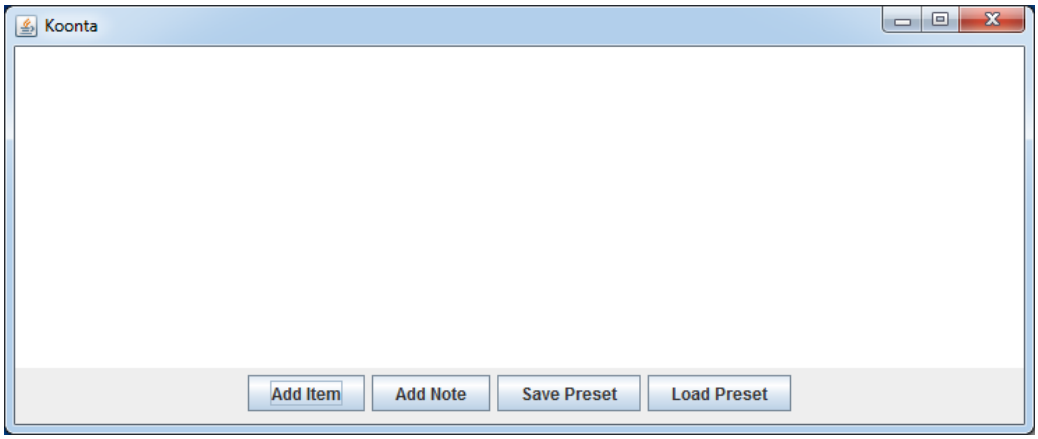
Ohjelman viimeisimmän version suunnitteluvaiheessa päädyttiin toiminnallisuusratkaisuun, jossa käyttäjä syöttää ohjelmaan valitsemansa kuvan verkko-osoitteen, jonka jälkeen kuva näkyy ohjelman päänäkymässä, on siirrettävissä sekä poistettavissa. Ohjelma toimii ainakin Jenkins CI:n sekä Agilefantin kanssa, sillä kuvat sijaitsevat palvelimilla staattisten verkko-osoitteiden takana. Sen sijaan, että ohjelma tekisi kyselyjä tietovarastopäivityksistä, ohjelma hakee kuvat uudelleen annetun osoitteen takaa kymmenen sekunnin välein välittämättä siitä, onko projekti tässä välissä koottu vai ei.

Sovellus yhdistetään palvelimelle syöttämällä sovellukseen sen palvelimen IP-osoite, jolle Jenkins CI on asennettu (Koontipalvelin, luku 2). Yhdistäessä palvelimelle annetaan myös portti, sekä protokolla, joka esimerkkikuvassa on https. Yhdistämisen yhteydessä kerrotaan ohjelmalle, että sen tulee luottaa kaikkiin sertifikaatteihin, jotka löytyvät annetun ip-osoitteen takaa. Kuvasta 18 on nähtävissä sovelluksen sisäänkirjautumisikkuna.



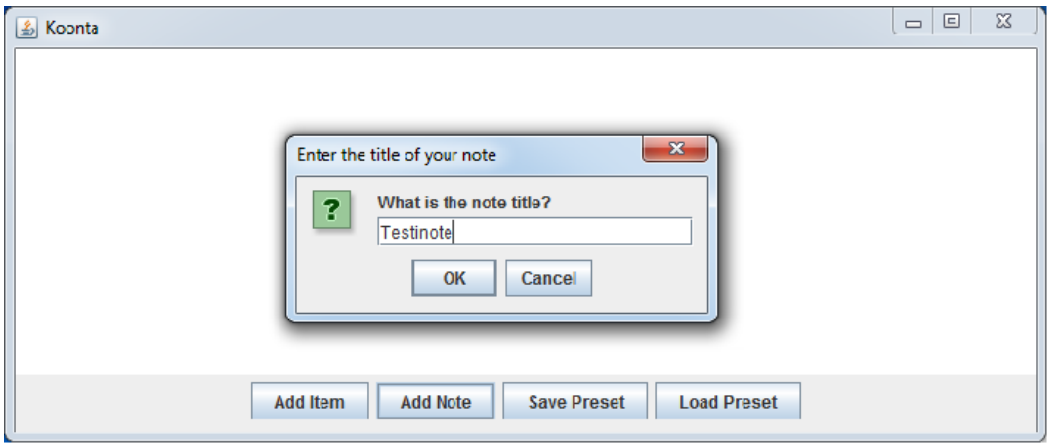
Kuva 18. Ohjelman sisäänkirjautumisikkuna.

Kun käyttäjä on kirjautunut palvelimelle, aukeaa sovelluksen päänäkymä (kuva 19).



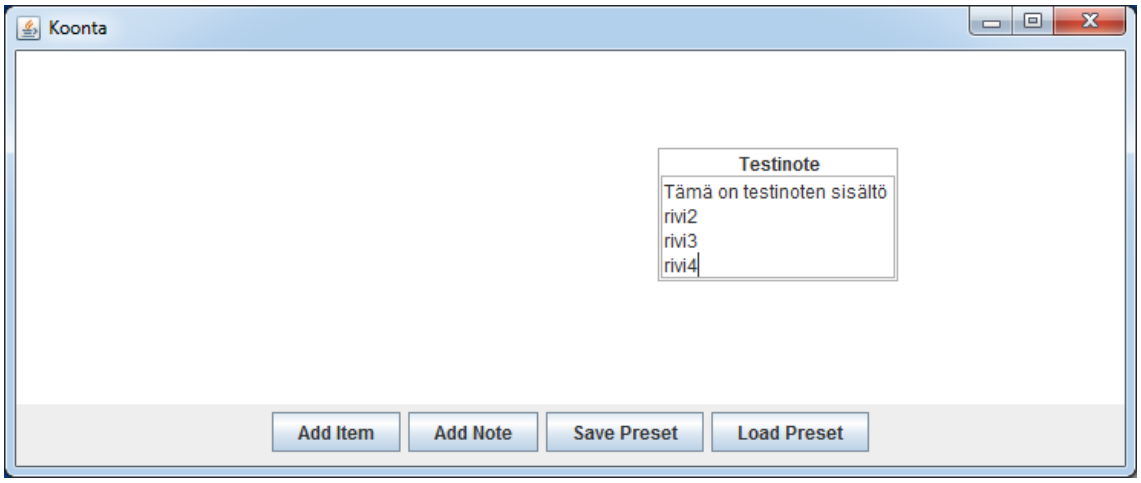
Kuva 19. Sovelluksen päänäkymä

Kuvasta 18 on nähtävissä sovelluksen toiminnallisuudelle tärkeät painikkeet: *Add Item, Add Note, Save Preset* ja *Load Preset.* Kun käyttäjä valitsee *Add Note*, kysyy ohjelma käyttäjältä *Note*:n otsikkoa (kuva 20).



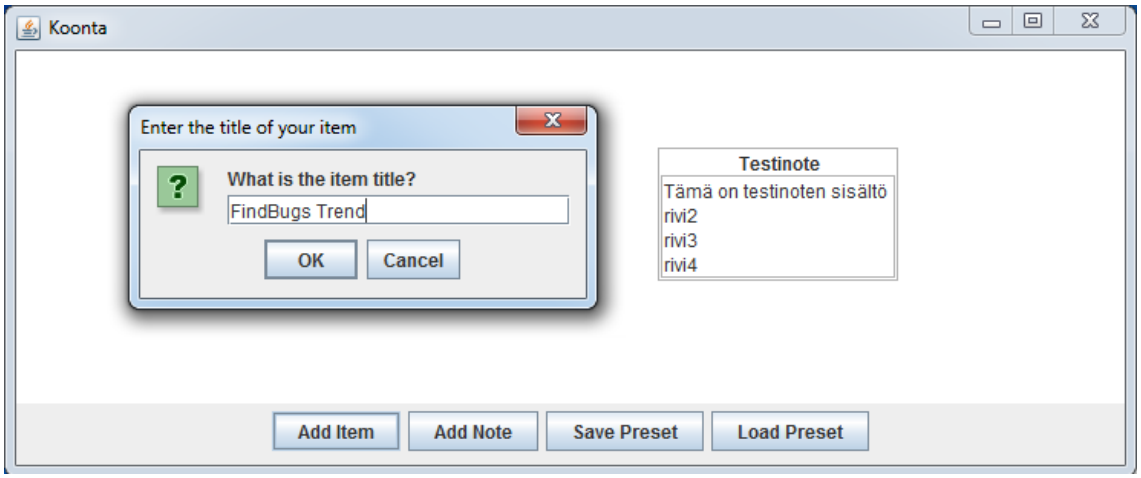
Kuva 20. Koonta pyytää käyttäjältä *Noten* otsikkoa

Kun muistilapulle (note) on annettu otsikko, ilmestyy sovelluksen päänäkymään muistilappu (kuva 21). Muistilapulle voi antaa haluamansa sisällön ja sen paikkaa voi vaihtaa raahaamalla sitä hiiren vasemmalla painikkeella otsikon kohdalta. Muistilapun leveyttä voi säätää hiiren rullalla, kun hiiren osoitinta pitää otsikon päällä.



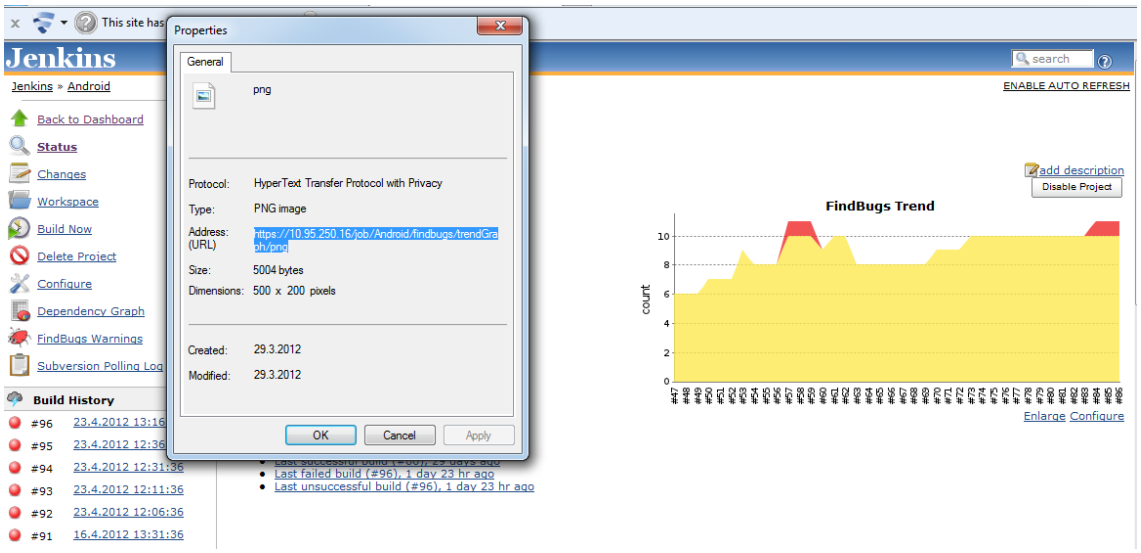
Kuva 21. Muistilappu sovelluksen päänäkymässä

Kun käyttäjä valitsee *Add Item* sovelluksen päänäkymästä, kysyy ohjelma käyttäjältä *Itemin* otsikkoa (kuva 22).



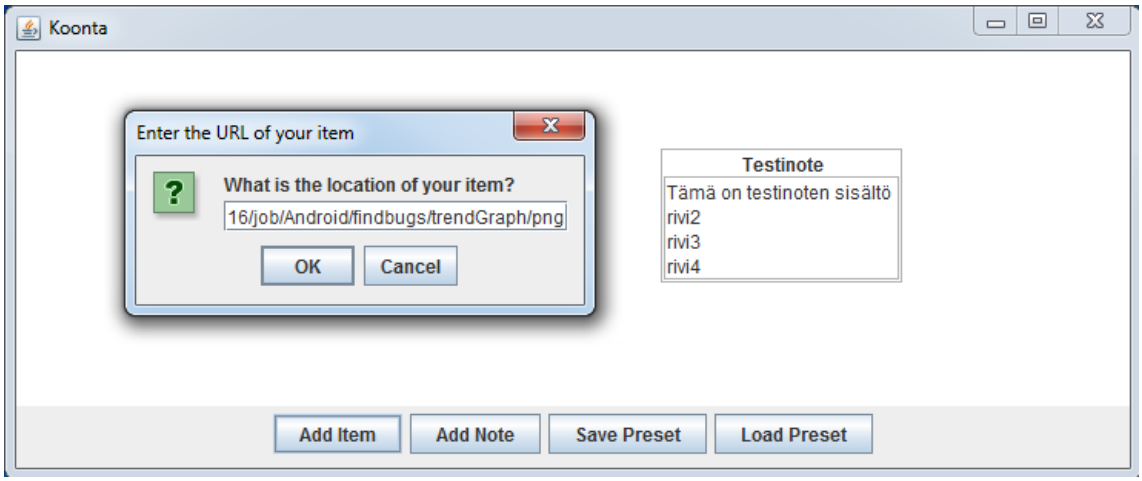
Kuva 22. Koonta pyytää käyttäjältä *Itemin* otsikkoa

Kun *Itemille* on annettu otsikko, on käyttäjän mentävä Jenkins CI web-käyttöliittymään ja valittava haluamansa kuva (kaavio). Kuvaa klikataan hiiren oikealla painikkeella ja kopioidaan kuvan URL-osoite leikepöydälle (kuva 23).



Kuva 23. Kuvan URL:in kopioiminen leikepöydälle

Kun URL-osoite on leikepöydällä, palataan Koonnan päänäkymään. Koonta pyytää käyttäjältä *Itemin* osoitetta ja leikepöydällä oleva URL kopioidaan Koontaan (kuva 24).

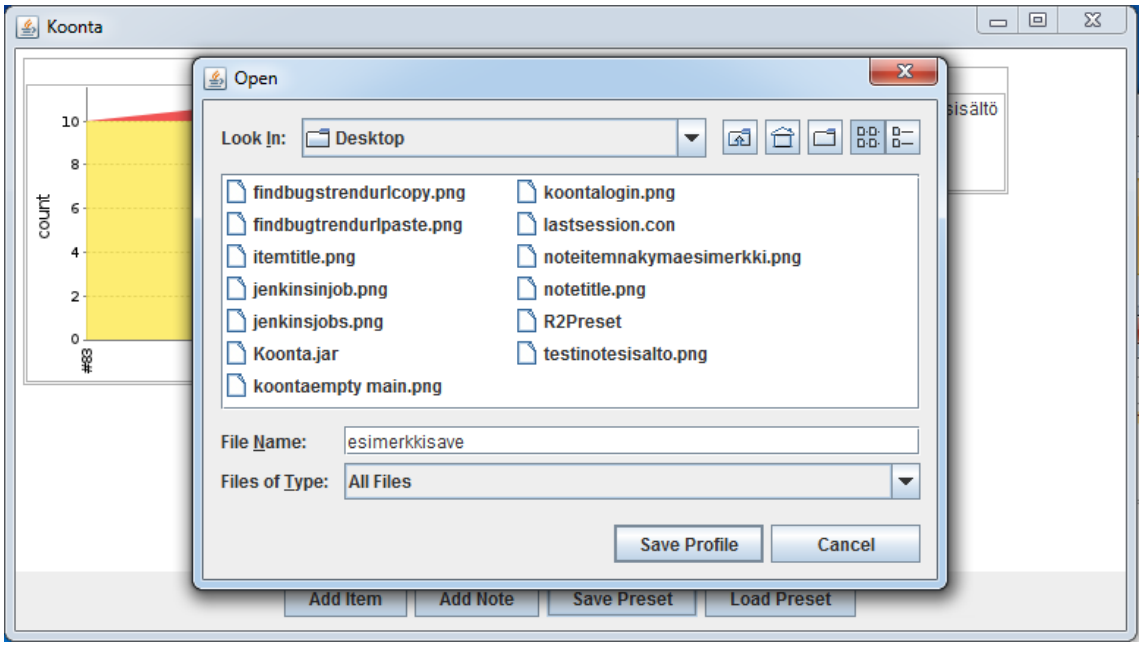


Kuva 24. *Itemin* kopioiminen leikepöydältä koontaan

Kun valitaan *OK*, sovelluksen päänäkymä muuttuu kuvan 25 mukaiseksi.

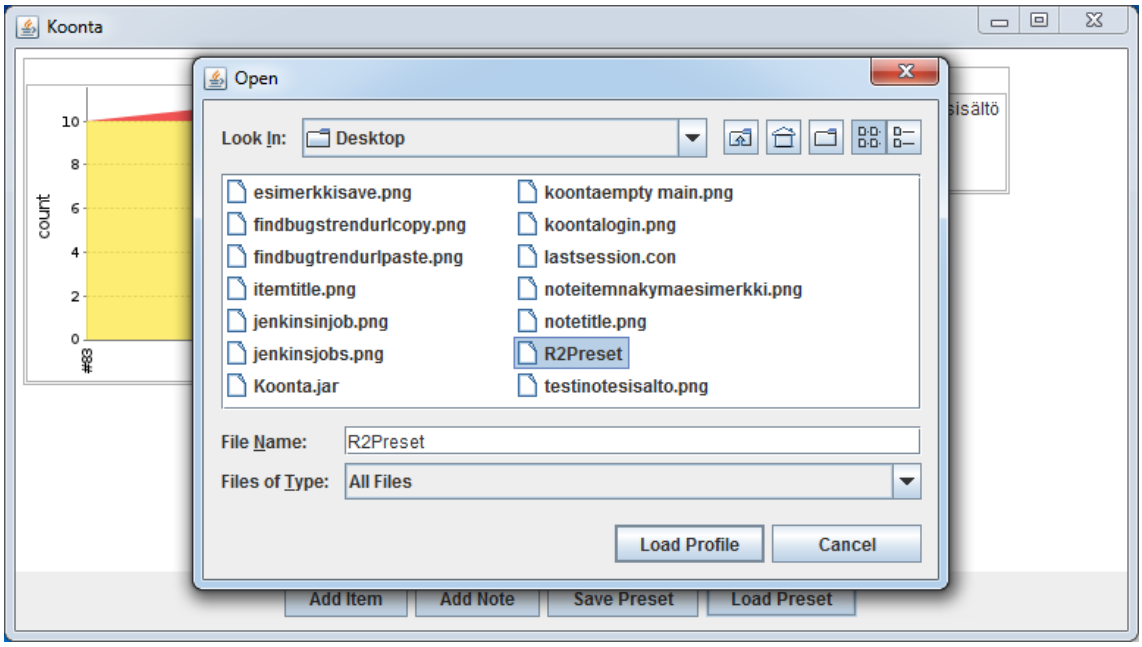
Kuva 25. Koonnan päänäkymä

Kun näkymään on liitetty kaikki halutut komponentit, valitaan päänäkymästä *Save Preset* (kuva 26).



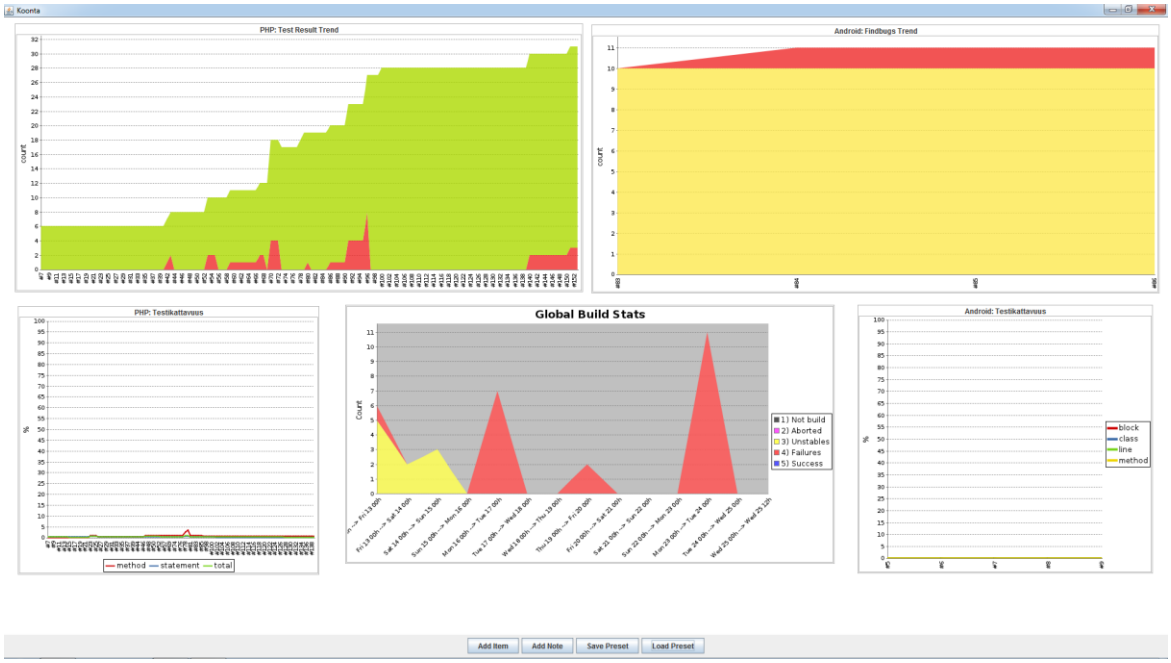
Kuva 26. Esiasetusten tallentaminen

Kun halutaan ladata aiemmin tallennettu esiasetus, valitaan päänäkymästä *Load Preset* (kuva 27).



Kuva 27. Esiasetusten lataaminen

Kuvassa ollaan lataamassa *R2Preset* -nimistä esiasetusta, joka on erään kevään 2012 Ohjelmistotuotantoprojekti-kurssin ryhmän esiasetus. Tämän ryhmän esiasetus näyttää kuvan 28 mukaiselta.



Kuva 28. Kevään 2012 kurssiryhmän esiasetus

Kuvasta on nähtävissä viisi eri *Item*-komponenttia: *PHP Test Result Trend, PHP Testikattavuus, Android FindBugs Trend, Android Testikattavuus* sekä *Global Build Stats.*

Jokainen *Item* pitää sisällään tiedon omasta URL-osoitteestaan sekä sijainnistaan näkymässä. Ohjelma pyyhkii näkymän 10 sekunnin välein, hakee kuvat uudestaan ja sijoittaa ne omille paikoilleen. Näin ollen näkymän kuvat päivittyvät oikeaan koontiin enintään 10 sekunnin viiveellä.

6.2 Sovelluksen kehittämisprosessi ja arkkitehtuuri

Tässä luvussa tutustutaan sovelluksen rakenteeseen ja toimintaperiaatteisiin. Lisäksi kerrotaan tuotteen kehittämisprosessista.

Projekti aloitettiin asentamalla työasemalle Virtual Box -niminen sovellus, jolle tehtiin koulun koontiympäristöä vastaava palvelinkonfiguraatio. Tätä palvelinta käytettiin sovelluksen toiminnallisuuden testaamiseen. Testipalvelimen käyttöjärjestelmänä toimii *Ubuntu 11.04 server x64*, joka on sama kuin kurssin koontipalvelimelle asennettu käyttöjärjestelmä. Tälle palvelimelle asennettiin Jenkins CI sekä muut tarvittavat sovellukset ohjelmiston testaamista varten.

Tuotteen suunnittelu aloitettiin käymällä läpi erilaisia käyttöliittymäideoita. Tämän jälkeen tehtiin erilaisia käyttöliittymän paperiprototyyppejä ja niistä valittiin sopivin. Kun käyttöliittymän ensimmäisen paperiversio valmistui, aloitettiin kirjoittaa käyttöliittymän luokkarakennetta ja valittiin sovelluksen käyttöliittymässä käytettävät Java-komponentit. Tässä vaiheessa se, mitä asiakas halusi, oli vielä osittain epäselvää. Ohjelmointi aloitettiin hieman liian aikaisin suunnitteluvaiheessa.

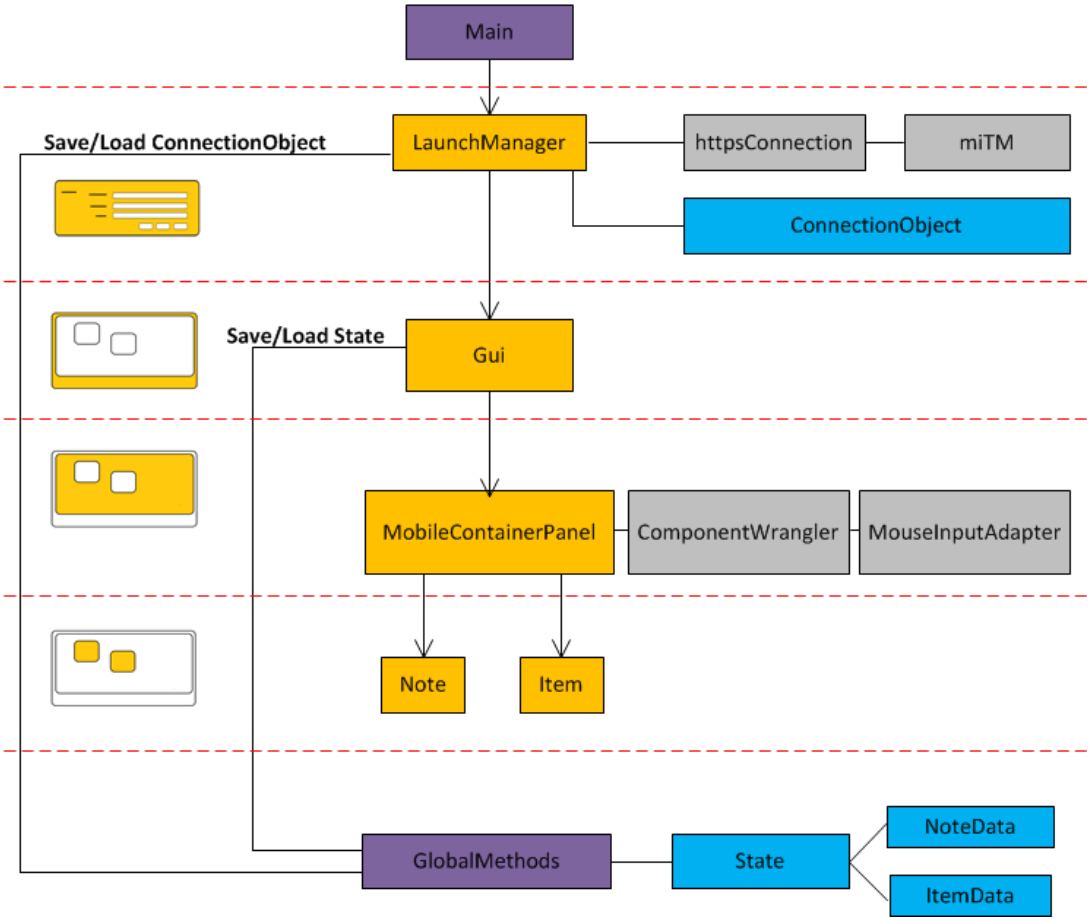
Ennen sovelluksen ohjelmoinnin aloittamista minulla oli erittäin vähän kokemusta tiedostojen käsittelystä, XML-parsimisesta, HTTP-yhteyden avaamisesta, sertifikaateista sekä Javan käyttöliittymäkomponenttien sijainnin ajon aikaisesta muuttamisesta. Tämän takia suunnitteluvaihetta ei voitu viedä loppuun asti ennen itse ohjelmoinnin aloittamista. Tästä seurasi se, että ohjelman luokkarakenne ei ole niin selkeä tai järkevä kuin se voisi olla. Ohjelmaan jäi lymyilemään pitkiksi ajoiksi myös ”kuolleita” luokkia, joiden toimintaperiaatteita ei jossain vaiheessa enää muistettu ja joita ei missään muussa luokassa tarvittu.

Ohjelmasta syntyi monta erilaista versiota erilaisilla käyttöliittymillä. Tällä hetkellä käytössä oleva ohjelma on ohjelman viides toiminnallinen versio. Ohjelma on vaihtanut kaksi kertaa luokkarakennetta ja viisi kertaa käyttöliittymää. Ohjelman kirjoittaminen aloitettiin kaksi kertaa kokonaan alusta.

Seuraavaksi esitellään ohjelman luokkakaavio ja jatketaan esittelemällä sovelluksen arkkitehtuurin ymmärtämisen kannalta välttämättömät luokat yksitellen, hierarkkisessa järjestyksessä, hierarkian huipulta aloittaen.

Kuvassa 29 on esitelty sovelluksen luokkakaavio. Kuvassa olevat oranssit luokat kuvaavat luokkia, jotka sisältävät graafisia komponentteja. Kaavion vasemmassa reunassa olevat pienet kuvat kuvaavat sovelluksen käyttöliittymää. Kuviin on merkitty luokan graafinen ilmentymä oranssilla värillä. Siniset luokat ovat tietorakenteita, joiden ilmentymiä voidaan tallentaa ja ladata. Harmaissa luokissa käsitellään osa käyttöliittymäluokkien toiminnallisuudesta, kuten yhteyden avaaminen koontipalvelimelle, sekä käyttöliittymäkomponenttien liikuttaminen ja päivittäminen päänäkymän piirtoalueella.

Main-luokka käynnistää ohjelman sisäänkirjautumisikkunan ja *GlobalMethods*-luokka sisältää ohjelman julkiset metodit, joita voidaan kutsua kaikista muista luokista. *GlobalMethods* sisältää julkisen ilmentymän tietorakenteesta *State,* joka voidaan tallentaa tiedostoon ja ladata tiedostosta. *State* sisältää tiedon *Item-* ja *Note*-olioiden graafisten ilmentymien sijainnista sekä koosta sovelluksen päänäkymässä. *GlobalMethods*-luokkaa käytetään myös *ConnectionObject*-tietorakenteen tallentamiseen. *ConnectionObject* tietue sisältää *LaunchManagerin* asetukset. Luokkien toiminnallisuudesta kerrotaan tarkemmin kuvan 29 jälkeen.



Kuva 29. Sovelluksen luokkakaavio.

Jatketaan esittelemällä ohjelman toiminnallisuuden kannalta tärkeimpien luokkien tarkoitus ja toiminnallisuus yksitellen:

Luokan nimi: GlobalMethods
Luokan tarkoitus ja toiminnallisuus: Sisältää julkisen ilmentymän tietueelle State. Osaa kirjoittaa minkä vain olion tiedostoon ja pakata tiedoston. Osaa purkaa tiedoston ja lukea olion puretusta tiedostosta. Luokan metodit ja attribuutit ovat julkisia kaikille luokille. Luokka luotiin alunperin siltä varalta, että sovellukseen kasaantuisi enemmän ns. irrallisia metodeita tai attribuutteja, joita tarvittaisiin usein ja monessa luokassa.

Luokan nimi: Main
Luokan tarkoitus ja toiminnallisuus: Käynnistää LaunchManager -näkymän parametrilla ConnectionObject. ConnectionObject -tietue pitää sisällään tiedon viime kirjau-

tumisesta. Mahdollistaa LaunchManagerin käynnistymisen viimeksi käytetyillä asetuksilla.

Luokan nimi: LaunchManager
Luokan tarkoitus ja toiminnallisuus: Käynnistää instanssin luokasta httpsConnection käyttäjän syöttämillä parametreilla. Käynnistää sovelluksen päänäkymäluokan Gui. Luokka tallentaa Gui:n käynnistyksen yhteydessä ConnectionObject -tietueen tiedostoon lastsession.con. Käyttäjä voi myös tallentaa asetukset manuaalisesti haluamaansa tiedostoon ja myöhemmin ladata asetukset haluamastaan tiedostosta.

Luokan nimi: httpsConnection
Luokan tarkoitus ja toiminnallisuus: Ottaa yhteyden Jenkins CI palvelimelle käyttäjän syöttämillä asetuksilla. Luo instanssin luokasta miTM.

Luokan nimi: miTM
Luokan tarkoitus ja toiminnallisuus: Toimii sovelluksen sertifikaatti-trust-managerina. Hyväksyy käyttäjän LaunchManagerissa syöttämien ip-osoitteiden takana olevat sertifikaatit.

Luokan nimi: Gui
Luokan tarkoitus ja toiminnallisuus: Toimii ohjelman päänäkymän kehyksenä. Sisältää komponentit kaavioiden ja muistilappujen lisäämiseen sekä esiasetusten tallentamiseen ja lataamiseen.

Luokan nimi: MobileContainerPanel
Luokan tarkoitus ja toiminnallisuus: Toimii paneelien Note ja Item piirtoalueena. Pitää sisällään listan ViewInterfacen toteuttavista olioista Note ja Item. Sisältää metodit esiasetusten tallentamiseen ja lataamiseen.

Luokan nimi: ComponentWrangler ja MouseInputAdapter
Luokan tarkoitus ja toiminnallisuus: On vastuussa käyttäjäinteraktiivista toiminnoista MobileContainerPaneelin sisällä. Kuvaajien sekä muistilappujen poisto hiiren oikealla painikkeella. Pitää huolen siitä, että kuvaajien ja muistilappujen sijainti tallentuu. On vastuussa myös kuvaajien ja muistilappujen drag'n'drop toiminnosta.

Luokan nimi: Item
Luokan tarkoitus ja toiminnallisuus: Item -olion tehtävä on näyttää käyttäjän valitsema kuva näytöllä. Sisältää ItemData –tietueen, joka pitää sisällään tiedon kuvan www-osoitteesta ja sen sijainnista MobileContainerPanel -kehyksen sisällä.

Luokan nimi: Note
Luokan tarkoitus ja toiminnallisuus: Note -olion tehtävä on näyttää käyttäjän lisäämä muistilappu näytöllä. Sisältää NoteData -tietueen, joka pitää sisällään tiedon muistilapun sisällöstä ja sen sijainnista MobileContainerPanel -kehyksen sisällä.

7 Yhteenveto

Tässä opinnäytetyössä tutustuttiin XF-laitteisiin, sekä XF-laitteiden pystyttämiseen ja niiden käytön kannalta välttämättömiin kehitysympäristökomponentteihin. Tarkoituksena oli kartoittaa erilaisia ratkaisuja informatiivisen työ- sekä kehitysympäristön rakentamiseen ja esitellä Metropolian ohjelmistotuotantoprojektissa työn kirjoittamisen aikana käytössä olleet työ- ja kehitysympäristöratkaisut.

Työssä esiteltiin kurssin uusi kehitys- sekä työympäristö ja tutustuttiin eri tapoihin tarkkailla koontiympäristöä. Koontiympäristön tarkkailu -käsitteen ymmärtämisen kannalta välttämätöntä oli tarpeellista tutustua myös kurssin kehitysympäristöön ja selventää lukijalle, mistä koontiprosessissa on kysymys.

Koonta sovelluksen suunnitteleminen ja ohjelmoiminen ei sujunut ongelmitta. Ohjelman oli tarkoitus toimia yhteen Agilefantin sekä Jenkins CI:n kanssa. Ohjelma ei kuitenkaan tue tällä hetkellä Agilefanttia ohjelmallisen kirjautumisongelman vuoksi, mutta tarkoituksena on ratkoa ongelma vielä tämän opinnäytetyön kirjoittamisen jälkeen.

Ohjelman suunnitteluvaihe jäi arkkitehtuuri- ja paperiprototyyppitasolle. Käyttöliittymän paperiprotoista toteutukseen käytettiin viimeisintä versiota. Arkkitehtuurisuunnitelmista ei kuitenkaan toteutunut yksikään. Olen sitä mieltä, että syksyn 2011 alussa ohjelmointitaitoni eivät riittäneet kunnolliseen ohjelmistoarkkitehtuurin suunnitteluun. Tästä seurasi sovelluksen epäjohdonmukainen rakenne. En usko tämän vaikuttaneen tuotteen lopputulokseen, vaan sen valmistumisnopeuteen.

Vaikka ohjelmiston suunnitteluvaihetta ei voi sanoa onnistuneeksi, eikä ohjelmaan ole implementoitu vielä kaikkia asiakkaan haluamia ominaisuuksia, voidaan sovelluksen lopputulokseen olla tyytyväisiä. Erilaisten hankaluuksien ja oman kokemattomuuteni vuoksi oli tämän projektin tekeminen oppimiskokemuksena korvaamaton.

Toivotaan, että tämä työ toimii hyvänä referaattina Metropolian ohjelmistotuotantoprojektin kehitysympäristön esittelemiseen ja antaa lukijalle hyvän yleiskuvan informatiivisen kehitysympäristön rakentamisesta sekä koonnin monitoroinnista.

Lähteet

Agilefant. Verkkosivu.
<http://www.agilefant.org/>. Luettu 14.2.2012.

Apache Ant. Verkkosivu.
<http://ant.apache.org/>. Luettu 15.4.2012.

Apache Maven Project. Verkkosivu.
<http://maven.apache.org/>. Luettu 16.4.2012.

A Simple Makefile Tutorial. Muokattu 2008. Verkkosivu.
<http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/>. Luettu 13.4.2012.

Bathie, Mark. 2010. Continuous Integration – Basic Overview and Best Practices. Verkkosivu. <http://blog.codesion.com/post/846607039/continuous-integration-overview-best-practice>. Luettu 10.5.2012.

Clark, Mike. 2004. Pragmatic Project Automation: How to Build, Deploy and Monitor Java Applications. The Pragmatic Programmers LLC. E-Kirja.
Ferguson John Smart. 2011. Jenkins the definitive guide: O'Reilly Media. E-kirja.

Jenkins Wiki. Verkkosivu. <https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins>.
Lukkarinen, Aleksi. 2011. Opinnäytetyö. Iteratiivinen sovelluskehitys: Kehitysympäristön toteutus ja ylläpito. Opinnäytetyö. Tietotekniikan koulutusohjelma. Metropolia Ammattikorkeakoulu. <http://urn.fi/URN:NBN:fi:amk-201105269796>.

Maven vs Ant or Ant vs Maven?. 1999-2006. Verkkojulkaisu.
<http://www.javafaq.nu/java-article1168.html>. Luettu 23.4.2012.

Sanchez, Carlos. 2009. Using lava lamps for continuous integration. Verkkojulkaisu.
<http://blog.carlossanchez.eu/2009/01/28/using-lava-lamps-for-continuous-in/>.
Luettu 12.4.2012.

Schneide Blog – public business secrets. Verkkosivu.
<http://schneide.wordpress.com/>. Luettu 12.4.2012.

Who broke the build? 2011. Verkkojulkaisu.
<http://www.papercut.com/blog/chris/2011/08/19/who-broke-the-build/>. Luettu 12.4.2011.